DALSA • 7075 Place Robert-Joncas, Suite 142 • St-Laurent, Quebec, H4M 2Z2 • Canada
http://www.imaging.com

Sapera++ LT™

Programmer's Manual

Edition 6.00

Part number OC-SAPM-SPPP0

NOTICE
© 2001-2006 DALSA Corp. All rights reserved.

This document may not be reproduced nor transmitted in any form or by any means, either electronic or mechanical, without the express written permission of DALSA Corp. Every effort is made to ensure the information in this manual is accurate and reliable. Use of the products described herein is understood to be at the user's risk. DALSA Corp. assumes no liability whatsoever for the use of the products detailed in this document and reserves the right to make changes in specifications at any time and without notice.

Microsoft is a registered trademark; Windows®, Windows NT®, Windows® 2000, and Windows® XP are trademarks of Microsoft® Corporation.
All other trademarks or intellectual property mentioned herein belong to their respective owners.

Printed on August 3, 2006

Document Number: OC-SAPM-SPPP0

Printed in Canada

Contents

INTRODUCTION 5
- OVERVIEW OF THE MANUAL 5
- ABOUT THE MANUAL 5
- USING THE MANUAL 5

GETTING STARTED 7
- WHAT IS SAPERA++? 7
- REQUIREMENTS 8
- FILE LOCATIONS 8

HIERARCHY CHARTS 9
- BASIC CLASS HIERARCHY CHART 9
- GUI CLASS HIERARCHY CHART 10

USING SAPERA++ 11
- HEADER FILES, LIBRARIES, AND DLLS 11
- CREATING A SAPERA++ APPLICATION 12
- DEMOS AND EXAMPLES 14

BASIC CLASS REFERENCE 15
- DATA CLASSES 15
- SAPACQUISITION 21
- SAPACQCALLBACKINFO 39
- SAPACQDEVICE 42
- SAPACQDEVICECALLBACKINFO 59
- SAPBAYER 66
- SAPBUFFER 80
- SAPBUFFERREMOTE 118
- SAPBUFFERROI 121
- SAPBUFFERWITHTRASH 127
- SAPCAB 131
- SAPCOUNTER 135
- SAPCOUNTERCALLBACKINFO 144
- SAPDISPLAY 147
- SAPDSP 157

SapFeature 160
SapFlatField 177
SapGio 193
SapGioCallbackInfo 204
SapGraphic 207
SapLocation 220
SapLut 223
SapManager 240
SapManCallbackInfo 260
SapPerformance 264
SapPixPro 266
SapPixProParams 273
SapProcessing 277
SapProCallbackInfo 285
SapTransfer 287
Specialized Transfer Classes 305
SapView 316
SapViewCallbackInfo 336
SapXferCallbackInfo 338
SapXferPair 342
SapXferParams 353
SapXferNode 358

GUI CLASS REFERENCE 363
CAbortDlg 363
CAcqConfigDlg 365
CAcqDlg 368
CAScanDlg 370
CBayerOptionsDlg 372
CBufDlg 374
CCabDlg 377
CCompDlg 380
CDspDlg 382
CFlatFieldDlg 385
CGioInputDlg 387
CGioOutputDlg 389
CImageWnd 391
CLoadSaveDlg 400
CLScanDlg 404
CPixProDlg 406
CViewDlg 409

DALSA CONTACT INFORMATION 411
Sales Information 411
Technical Support 412

GLOSSARY OF TERMS 413

INDEX 415

Introduction

Overview of the Manual

The *Sapera++ LT Programmer's* Manual covers the following topics:

- Getting Started
- Using Sapera++
- Basic Class Reference
- GUI Class Reference
- Hierarchy Charts
- DALSA Contact Information

About the Manual

This manual exists in printed, Windows Help, and Adobe Acrobat (PDF) formats. The Help and PDF formats make full use of hypertext cross-references. The PDF format offers links to DALSA's home page on the Internet, located at **http://www.imaging.com**.

DALSA Web site contains documents, updates, demos, errata, utilities, and more.

Using the Manual

File names, directories, and Internet sites will be in bold text (for example, **setup.exe**, **c:\windows**, **http://www.imaging.com**). Function parameters will be in italics (for example, *xlen*).

Source code, code examples, text file listings, and text that must be entered using the keyboard will be in typewriter-style text (for example, `[Pixel Clock]`).

Menu and dialog actions will be indicated in bold text in the order of the instructions to be executed, with each instruction separated by bullets. For example, going to the **File** menu and choosing **Save** would be written as **File•Save**.

Getting Started

What is Sapera++?

Sapera++ is the main Application Programming Interface (API) for Sapera LT. It is a C++ class library that encapsulates the low-level Sapera C Library, providing most of its functionality. It also provides some features not found in the latter, for example, Windows GDI graphics in non-destructive overlay on top of an image. Sapera++ provides high-level classes reducing application code complexity. The Sapera++ architecture reflects the underlying Sapera architecture. This provides the user with a high-level of flexibility while keeping the simplicity and compactness of object-oriented code. Sapera++ contains two sets of classes: Basic and the GUI.

The *Basic Classes* are a set of C++ classes containing commonly used Sapera code that can be used with many imaging applications. These classes are completely user-interface independent. They address the basic concepts of imaging applications, such as Acquisition Transfers, Processing, and Display. The *Basic Classes* main purpose is to simplify application code by reducing considerably the number of calls to Sapera functions and parameters. Source code is no longer included for these classes with Sapera LT, only the header files and libraries necessary for application development are available.

The *GUI Classes* are an additional set of C++ classes containing commonly used dialog boxes and windows used in many imaging applications. These classes are MFC™ dependent. They are intended to be included in any MFC-based imaging application in order to improve rapid-application development (RAD). GUI class source code is included with Sapera LT so that they may be freely modified. Sapera++ applications do not have to use these classes, they may implement their own dialog boxes and windows instead.

Requirements

Sapera++ currently supports the following compilers:

- Microsoft Visual C++ 6.0
- Microsoft Visual C++ .NET 2003
- Microsoft Visual C++ .2005
- Borland C++ Builder, version 6 (*Basic Classes* only)

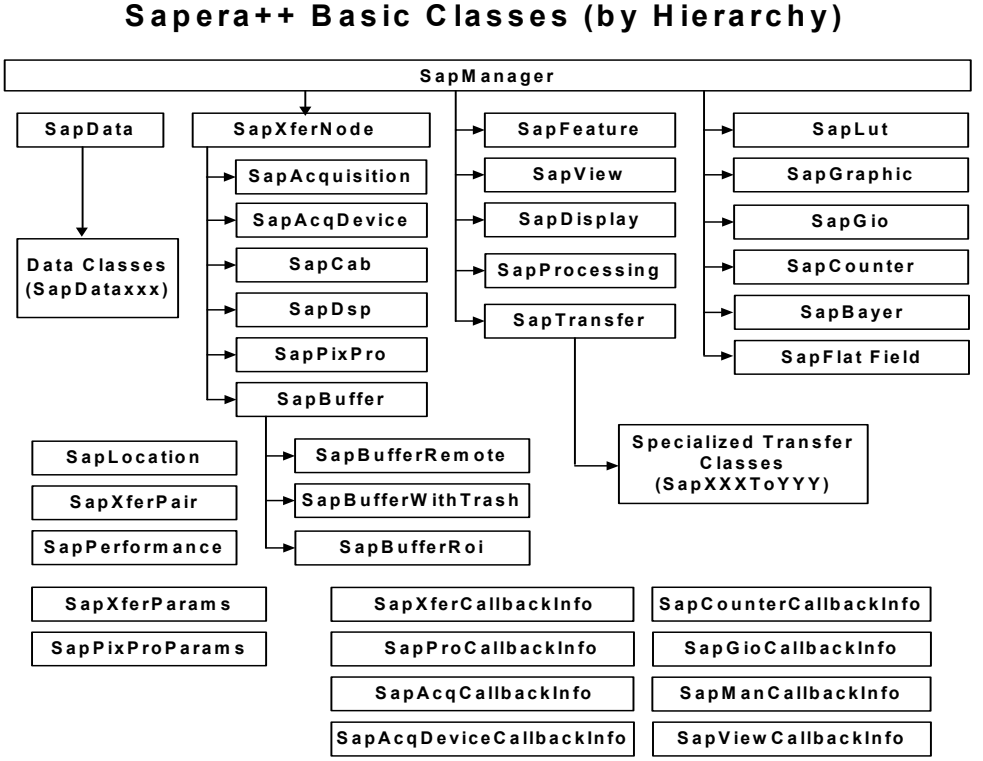
File Locations

The table below shows the different file groups with their respective locations.

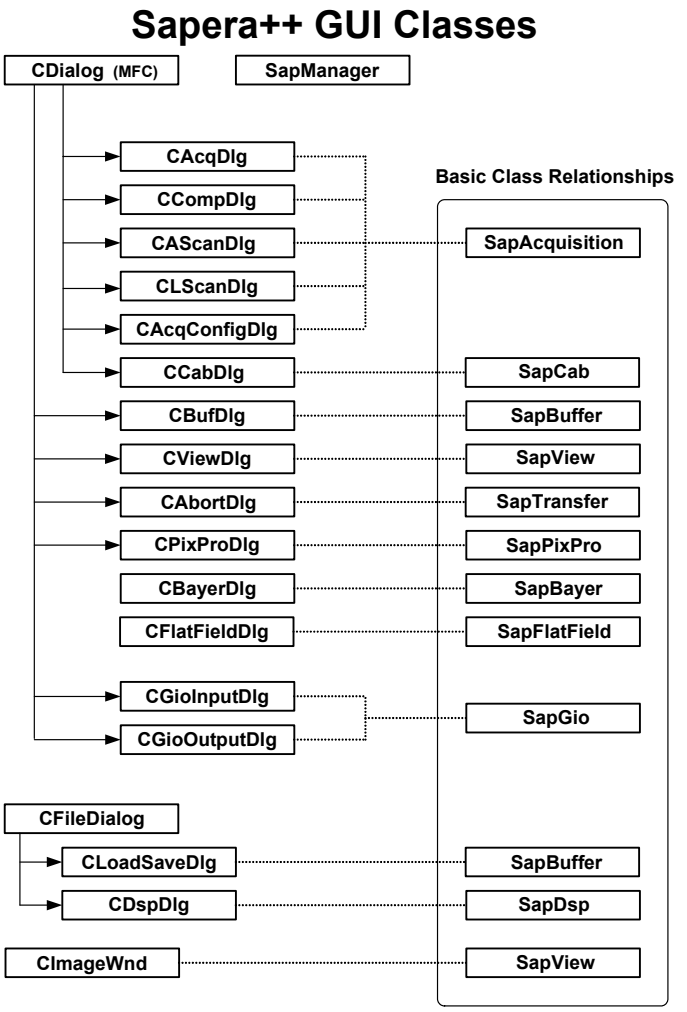
Description	Location
Basic Classes headers	**Sapera\Classes\Basic**
GUI Classes headers and source code	**Sapera\Classes\Gui**
Import librairies	**Sapera\Lib**
Import librairies (Visual C++ .NET 2003)	**Sapera\Lib\.NET_2003**
Import libraries (C++ Builder 6)	**Sapera\Lib\Bc**
Dynamic-link libraries (DLLs)	Windows system directory (**<windir>\System32**)

Hierarchy Charts

Basic Class Hierarchy Chart



GUI Class Hierarchy Chart



Using Sapera++

Header Files, Libraries, and DLLs

The following files are provided with Sapera LT. Note that library and DLL files with the 'D' suffix (for example, **SapClassBasicD.lib**) correspond to debug versions of files with the same name but without the suffix. Also, 'XX' refers to the current Sapera LT version number, for example, **SapClassBasic52.dll** for the version 5.20 Basic Classes DLL.

File Name	Description	Location
SapClassBasic.h	Basic class header file	**Sapera\Classes\Basic**
SapClassGui.h	GUI class header file	**Sapera\Classes\Gui**
SapClassBasic.lib **SapClassBasicD.lib**	Basic class libraries for Visual C++ 6.0 and Visual C++ .NET	**Sapera\Lib**
SapClassGui.lib **SapClassGuiD.lib**	GUI class libraries for Visual C++ 6.0	**Sapera\Lib**
SapClassGui.lib **SapClassGuiD.lib**	GUI class libraries for Visual C++ .NET 2003	**Sapera\Lib\.NET_2003**
SapClassBasic.lib	Basic class library for C++ Builder 6	**Sapera\Lib\Bc**
SapClassBasicXX.dll **SapClassBasicXXD.dll**	Basic class DLLs	**<windir>\System32**
SapClassGuiXX.dll **SapClassGuiXXD.dll**	GUI class DLLs	**<windir>\System32**
SapClassGuiXX.NET_2003.dll **SapClassGuiXXD.NET_2003.dll**	GUI class DLLs for Visual C++ .NET 2003	**<windir>\System32**
SapClassBasicXX_b.dll	Basic class DLL for C++ Builder 6	**<windir>\System32**

Creating a Sapera++ Application

Directions to create a Sapera++ application in Visual Studio 6.0, Visual Studio .NET 2003, Visual Studio 2005, and C++ Builder 6.

Visual Studio 6.0

Follow the steps below to compile and link an application that uses the *Basic Classes*:

1. Include **SapClassBasic.h** in the program source code (it includes all other required headers)
2. Add **$(SAPERADIR)\Classes\Basic** in *Project* | *Settings…* | *C/C++* | *Preprocessor* | *Additional include directories*
3. Insert **$(SAPERADIR)\Lib\SapClassBasic.lib** and **SapClassBasicD.lib** in *Project* | *Add to Project* | *Files*
4. In *Project* | *Settings…*| *General* for **SapClassBasic.lib**, select *Exclude file from build* for *Win32 Debug*
5. In *Project* | *Settings…*| *General* for **SapClassBasicD.lib**, select *Exclude file from build* for *Win32 Release*
6. In *Project* | *Settings…* | *C/C++* | *Code Generation* | *Use run-time library*, choose the option *Multithreaded DLL* (in release mode) or *Debug Multithreaded DLL* (in debug mode)

Follow the steps below to compile and link an application that uses the *GUI Classes*:

1. Include **SapClassGui.h** in the program source code (it includes all other required headers)
2. Add **$(SAPERADIR)\Classes\Gui** in *Project* | *Settings…* | *C/C++* | *Preprocessor* | *Additional include directories*
3. Insert **$(SAPERADIR)\Lib\SapClassGui.lib** and **SapClassGuiD.lib** in *Project* | *Add to Project* | *Files*
4. In *Project* | *Settings…*| *General* for **SapClassGui.lib**, select *Exclude file from build* for *Win32 Debug*
5. In *Project* | *Settings…*| *General* for **SapClassGuiD.lib**, select *Exclude file from build* for *Win32 Release*
6. In *Project* | *Settings…* | *C/C++* | *Code Generation* | *Use runtime library*, choose the option *Multithreaded DLL* (in release mode) or *Debug Multithreaded DLL* (in debug mode)

Visual Studio .NET 2003

Follow the steps below to compile and link an application that uses the *Basic Classes*:

1. Include **SapClassBasic.h** in the program source code (it includes all other required headers)
2. Add **$(SAPERADIR)\Classes\Basic** in *Project* | *Properties* | *C/C++* | *General* | *Additional Include Directories*
3. Insert **$(SAPERADIR)\Lib\SapClassBasic.lib** and **SapClassBasicD.lib** in *Project* | *Add Existing Item ...*
4. In *Project* | *Properties* | *General* for **SapClassBasic.lib**, select *Excluded From Build* for *Debug*
5. In *Project* | *Properties* | *General* for **SapClassBasicD.lib**, select *Excluded From Build* for *Release*
6. In *Project* | *Properties* | *C/C++* | *Code Generation* | *Runtime Library*, choose the option *Multi-threaded* DLL (in release mode) or *Multi-threaded Debug* DLL (in debug mode)

Follow the steps below to compile and link an application that uses the *GUI Classes*:

1. Include **SapClassGui.h** in the program source code (it includes all other required headers)
2. Add **$(SAPERADIR)\Classes\Gui** in *Project* | *Properties* | *C/C++* | *General* | *Additional Include Directories*
3. Insert **$(SAPERADIR)\Lib\.NET_2003\SapClassGui.lib** and **SapClassGuiD.lib** in *Project* | *Add Existing Item ...*
4. In *Project* | *Properties* | *General* for **SapClassGui.lib**, select *Excluded From Build* for *Debug*
5. In *Project* | *Properties* | *General* for **SapClassGuiD.lib**, select *Excluded From Build* for *Release*
6. In *Project* | *Properties* | *C/C++* | *Code Generation* | *Runtime Library*, choose the option *Multi-threaded* DLL (in release mode) or *Multi-threaded Debug* DLL (in debug mode)

Visual Studio 2005

Follow the same steps as for Visual Studio .NET 2003 to compile and link an application that uses the *Basic Classes*:

Follow the steps below to compile and link an application that uses the *GUI Classes*:

1. In the Sapera installation directory, open the solution Demos\Classes\Vc\SapDemos_2005.sln
2. Rebuild the SapClassGui project for *Debug* and *Release*
3. Include **SapClassGui.h** in the program source code (it includes all other required headers)
4. Add **$(SAPERADIR)\Classes\Gui** in *Project* | *Properties* | *C/C++* | *General* | *Additional Include Directories*
5. Insert **$(SAPERADIR)\Lib\.NET_2005\SapClassGui.lib** and **SapClassGuiD.lib** in *Project* | *Add Existing Item …*
6. In *Project* | *Properties* | *General* for **SapClassGui.lib**, select *Excluded From Build* for *Debug*
7. In *Project* | *Properties* | *General* for **SapClassGuiD.lib**, select *Excluded From Build* for *Release*
8. In *Project* | *Properties* | *C/C++* | *Code Generation* | *Runtime Library*, choose the option *Multi-threaded* DLL (in release mode) or *Multi-threaded Debug* DLL (in debug mode)

C++ Builder 6

Follow the steps below to compile and link an application that uses the *Basic Classes*:

1. Include **SapClassBasic.h** in the program source code (it includes all other required headers)
2. Add **$(SAPERADIR)\Classes\Basic** in *Project* | *Options…* | *Directories/Conditionals* | *Include path*
3. Insert **$(SAPERADIR)\Lib\Bc\SapClassBasic.lib** in *Project* | *Add to Project …*

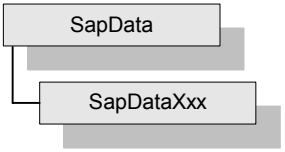
The *GUI Classes* are not supported under C++ Builder

Demos and Examples

Refer to the *Sapera LT User's Manual* for a description of the Sapera++ demos as well as examples available in Sapera LT.

Basic Class Reference

Data Classes



SapData and its derived classes act as wrappers for low-level Sapera LT data types, where each class encapsulates one data element of a specific type. They are used as method arguments or return values in various Sapera++ classes.

#include <SapClassBasic.h>

Basic Class Hierarchy Chart

SapData Class

Purpose
This is the common base class for all other data classes. Though SapData objects may be directly instantiated, they serve no useful purpose.

SapData();
Class constructor

virtual void **Clear**();
Clears the data element to black, which may not correspond to the numeric value 0.

SapFormatType **GetType**();
Identifies to which SapDataXxx class the current object is an instance. See the SapManager::GetFormatType method for the list of available types.

Data Classes | Basic Class Hierarchy Chart

SapDataMono Class

Purpose
Encapsulates one element supporting Sapera monochrome data types (excluding 64-bit)

SapDataMono();
SapDataMono(int *mono*);
Class constructor, where the *mono* argument specifies an initial value other than black

int **Mono**();
Returns the current value of the data element

void **Set**(int *mono*);
Specifies a new value for the data element

Data Classes | Basic Class Hierarchy Chart

SapDataRGB Class

Purpose
Encapsulates one element supporting Sapera RGB data types

SapDataRGB();
SapDataRGB(int *red*, int *green*, int *blue*);
Class constructor, where the *red*, *green*, and *blue* arguments specify an initial value other than black

int **Red**();
Returns the red component of the current value of the data element

int **Green**();
Returns the green component of the current value of the data element

int **Blue**();
Returns the blue component of the current value of the data element

void **Set**(int *red*, int *green*, int *blue*);
Specifies a new value for the data element

Data Classes | Basic Class Hierarchy Chart

SapDataRGBA Class

Purpose
Encapsulates one element supporting Sapera RGB with alpha channel data types

SapDataRGBA();SapDataRGBA(int *red*, int *green*, int *blue*, int *alpha*);
Class constructor, where the *red*, *green*, *blue* and *alpha* arguments specify an initial value other than black

int Red();
Returns the red component of the current value of the data element

int Green();
Returns the green component of the current value of the data element

int Blue();
Returns the blue component of the current value of the data element

int Alpha();
Returns the alpha component of the current value of the data element

void Set(int *red*, int *green*, int *blue*, int *alpha*);
Specifies a new value for the data element

Data Classes | Basic Class Hierarchy Chart

SapDataYUV Class

Purpose
Encapsulates one element supporting Sapera YUV data types

SapDataYUV();
SapDataYUV(int *y*, int *u*, int *v*);
Class constructor, where the *y*, *u*, and *v* arguments specify an initial value other than black

int **Y**();
Returns the Y component of the current value of the data element

int **U**();
Returns the U component of the current value of the data element

int **V**();
Returns the V component of the current value of the data element

void **Set**(int *y*, int *u*, int *v*);
Specifies a new value for the data element

Data Classes | Basic Class Hierarchy Chart

SapDataHSI Class

Purpose
Encapsulates one element supporting Sapera HSI data types

SapDataHSI();
SapDataHSI(int *h*, int *s*, int *i*);
Class constructor, where the *h*, *s*, and *i* arguments specify an initial value other than black

int **H**();
Returns the H component of the current value of the data element

int **S**();
Returns the S component of the current value of the data element

int **I**();
Returns the I component of the current value of the data element

void **Set**(int *h*, int *s*, int *i*);
Specifies a new value for the data element

Data Classes | Basic Class Hierarchy Chart

SapDataHSV Class

Purpose
Encapsulates one element supporting Sapera HSV data types

SapDataHSV();
SapDataHSV(int *h*, int *s*, int *v*);
Class constructor, where the *h*, *s*, and *v* arguments specify an initial value other than black

int **H**();
Returns the H component of the current value of the data element

int **S**();
Returns the S component of the current value of the data element

int **V**();
Returns the V component of the current value of the data element

void **Set**(int *h*, int *s*, int *v*);
Specifies a new value for the data element

Data Classes | Basic Class Hierarchy Chart

SapDataPoint Class

Purpose
Encapsulates one element supporting Sapera data types representing integer (x, y) coordinate pairs

SapDataPoint();
SapDataPoint(int *x*, int *y*);
Class constructor, where the *x* and *y* arguments specify an initial value other than (0, 0)

int **X()**;
Returns the X component of the current value of the data element

int **Y**();
Returns the Y component of the current value of the data element

void **Set**(int *x*, int *y*);
Specifies a new value for the data element

Data Classes | Basic Class Hierarchy Chart

SapDataFloat Class

Purpose
Encapsulates one element supporting Sapera floating-point data types

SapDataFloat();
SapDataFloat(float *flt*);
Class constructor, where the *flt* argument specifies an initial value other than black

int **Float**();
Returns the current value of the data element

void **Set**(float *flt*);
Specifies a new value for the data element

Data Classes | Basic Class Hierarchy Chart

SapDataFPoint Class

Purpose
Encapsulates one element supporting Sapera data types representing floating-point (x, y) coordinate pairs

SapDataFPoint();
SapDataFPoint(float *x*, float *y*);
Class constructor, where the *x* and *y* arguments specify an initial value other than (0.0, 0.0)

float **X()**;
Returns the X component of the current value of the data element

float **Y**();
Returns the Y component of the current value of the data element

void **Set**(float *x*, float *y*);
Specifies a new value for the data element

Data Classes | Basic Class Hierarchy Chart

SapDataFRGB Class

Purpose
Encapsulates one element supporting Sapera floating-point RGB data types

SapDataRGB();
SapDataRGB(float *red*, float *green*, float *blue*);
Class constructor, where the *red*, *green*, and *blue* arguments specifies an initial value other than black

float **Red**();
Returns the red component of the current value of the data element

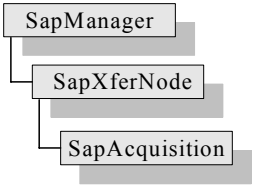
float **Green**();
Returns the green component of the current value of the data element

float **Blue**();
Returns the blue component of the current value of the data element

void **Set**(float *red*, float *green*, float *blue*);
Specifies a new value for the data element

Data Classes | Basic Class Hierarchy Chart

SapAcquisition



The SapAcquisition Class includes the functionality to manipulate an acquisition resource. It is used as a source transfer node to allow data transfers from an acquisition resource to another transfer node, such as buffer, CAB, DSP, or Pixel Processor.

Note: GigE-Vision devices are not supported by this class. The *SapAcqDevice* class must be used in such cases.

#include <SapClassBasic.h>

SapAcquisition Class Members | Base Class: SapManager | Basic Class Hierarchy Chart

SapAcquisition Class Members

Construction	
SapAcquisition	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Attributes	
GetConfigFile, SetConfigFile	Gets/sets the name of the acquisition configuration file (CCF)
GetLabel	Gets a text description of the acquisition resource
GetEventType SetEventType	Gets/sets the combination of registered acquisition event types
SetCallbackInfo	Sets the application callback method for acquisition events and the associated context
GetCallback	Gets the current application callback method for acquisition events

GetContext	Gets the application context associated with acquisition events
GetCamSel SetCamSel	Gets/sets the current camera selector value
IsLutEnabled	Gets the current LUT enable value
CanEnableLut	Checks if the acquisition lookup table may be enabled/disabled
GetFlipMode SetFlipMode	Gets/sets the flipping (that is, mirroring) mode for acquired images
GetNumPlanarInputs	Gets the number of cameras used for acquiring into vertical planar buffers
GetPlanarInputs, SetPlanarInputs	Gets/sets the current configuration for acquiring into vertical planar buffers
IsFlatFieldAvailable	Gets availability of hardware-based flat-field correction
IsBayerAvailable	Gets availability of hardware-based Bayer conversion
GetSerialPortName	Gets the name of the serial port attached to the current acquisition device
Operations	
SaveParameters	Saves the acquisition parameters to an acquisition configuration file (CCF)
GetLut	Gets the current acquisition lookup table
ApplyLut	Programs a new acquisition lookup table
EnableLut	Enables/disables the acquisition lookup table
IsSignalStatusAvailable	Checks for availability of the status of input acquisition signals
GetSignalStatus	Gets the current status of input acquisition signals
SoftwareTrigger	Simulates a trigger to the acquisition device
IsCapabilityValid	Checks for the availability of a low-level Sapera C library capability
IsParameterValid	Checks for the availability of a low-level Sapera C library parameter
GetCapability	Gets the value of a low-level Sapera C library capability
GetParameter SetParameter	Gets/sets the value of a low-level Sapera C library parameter
CustomCommand	Issues a low-level custom command specific to the acquisition hardware

SapAcquisition | SapManager | Basic Class Hierarchy Chart

Member Functions

The following functions are members of the SapAcquisition Class.

SapAcquisition::SapAcquisition

```
SapAcquisition (
  SapLocation loc = SapLocation::ServerSystem,
);
SapAcquisition (
  SapLocation loc,
  const char *configFile,
  SapAcquisition::EventType eventType = SapAcquisition::EventNone,
  SapAcqCallback pCallback = NULL
  void *pContext = NULL
);
SapAcquisition (
  SapLocation loc,
  const char *camfile,
  const char *vicfile,
  SapAcquisition::EventType eventType = SapAcquisition::EventNone,
  SapAcqCallback pCallback = NULL
  void *pContext = NULL
);
SapAcquisition (
  const SapAcquisition &acq,
  SapAcquisition::EventType eventType,
  SapAcqCallback pCallback
  void *pContext = NULL
);
```

Parameters

loc	SapLocation object specifying the server where the acquisition resource is located and the index of the acquisition resource on this server.
configFile	Name of the acquisition configuration file (CCF) that describes all camera and frame grabber-related acquisition parameters. Use one of the standard CCF files provided with Sapera or create one using the CamExpert utility.
camfile	Name of the configuration file (CCA) that describes all camera related acquisition parameters (obsolete)

vicfile Name of the configuration file (CVI) that describes all frame grabber-related acquisition parameters (obsolete)

eventType Acquisition events for which the application callback function will be called. One or more of the following values may be combined together using a bitwise OR operation:

SapAcquisition::EventNone	No events
SapAcquisition::EventStartOfFrame	Start of frame
SapAcquisition::EventStartOfField	Start of any field (odd or even)
SapAcquisition::EventStartOfOdd	Start of odd field
SapAcquisition::EventStartOfEven	Start of even field
SapAcquisition::EventEndOfFrame	End of frame
SapAcquisition::EventEndOfField	End of any field (odd or even)
SapAcquisition::EventEndOfOdd	End of odd field
SapAcquisition::EventEndOfEven	End of even field
SapAcquisition::EventEndOfNLines	After a specific line number *eventType* = SapAcquisition::EventEndOfNLines | *numLines*
SapAcquisition::EventVirtualFrame	Equivalent to EventStartOfFrame for linescan cameras
SapAcquisition::EventExternalTrigger	Received an external trigger that will then acquire at least one image. The maximum callback rate cannot be greater than the acquisition video frame rate.
SapAcquisition::EventVerticalSync	Vertical sync detected, even if not acquiring
SapAcquisition::EventNoPixelClk	No pixel clock detected. Generated only once, unless a new SapTransfer::Snap/Grab command is issued or the pixel clock is detected again and then lost.
SapAcquisition::EventPixelClk	Pixel clock detected. Generated only once, unless a new SapTransfer::Snap/Grab command is issued or the pixel clock is lost again and then detected.
SapAcquisition::EventFrameLost	Lost a frame during live acquisition. This usually occurs if there is not enough bandwidth to transfer images to host memory.
SapAcquisition::EventDataOverflow	Data overflow occurred during live acquisition. This usually occurs if the acquisition device cannot sustain the data rate of the incoming images.
SapAcquisition:: EventExternalTriggerIgnored	Dropped an external trigger event. This usually occurs when the external trigger rate is faster then the acquisition frame rate.

SapAcquisition:: EventExternalTriggerTooSlow	The detected external trigger rate is too slow for the hardware to process. This can usually occur when using the shaft encoder multiplier.
SapAcquisition::EventHsyncLock	Detected a horizontal sync unlock to lock condition.
SapAcquisition::EventHsyncUnlock	Detected a horizontal sync lock to unlock condition.
SapAcquisition::EventVerticalTimeout	Detected a vertical timeout. You can set the timeout value by calling the SetParameter method for CORACQ_PRM_VERTICAL_TIMEOUT_DELAY.

Important Note: You will not usually need to catch acquisition events. They must not be confused with the transfer event mechanism used in almost all applications. If you need acquisition events, review the User's Manual for your acquisition hardware to find which ones are supported. For transfer related events, see the SapTransfer Class for more information.

pCallback Application callback function to be called each time one of the events specified above is received. If *eventType* is EventNone, this parameter is ignored.

The callback function must be declared as:

void MyCallback(SapAcqCallbackInfo **pInfo*);

pContext Optional pointer to an application context to be passed to the callback function. If *pCallback* is NULL or *eventType* is EventNone, this parameter is ignored.

acq Existing acquisition object

Remarks

The SapAcquisition constructor does not actually create the low-level Sapera resources. To do this, you must call the Create method.

The constructor with the *camFile* and *vicFile* arguments is now obsolete. However, you may use it for backward compatibility with older versions of Sapera LT in which CCA and CVI files were used instead of CCF files.

The constructor with an acquisition object, event type, callback function and context is useful in one particular case. If you use the GUI class CAcqConfigDlg to load a configuration file, the resulting acquisition object is not configured to handle events. You can then use this constructor to complete the configuration.

The SapAcquisition object is used only for storing the acquisition resource parameters. To acquire data, use the SapTransfer Class (or one of its derived classes) and pass the SapAcquisition object as a parameter for the constructor. SapTransfer then handles the actual data transfer.

SapAcquisition | SapAcquisition Class Members | Basic Class Hierarchy Chart

SapAcquisition::Create

virtual BOOL **Create**();

Return Value

Returns TRUE if the object was successfully created, FALSE otherwise

Remarks

Creates all the low-level Sapera resources needed by the acquisition object. Always call this method before SapTransfer::Create.

SapAcquisition | SapAcquisition Class Members | Basic Class Hierarchy Chart

SapAcquisition::Destroy

virtual BOOL **Destroy**();

Return Value

Returns TRUE if the object was successfully destroyed, FALSE otherwise

Remarks

Destroys all the low-level Sapera resources needed by the acquisition object. Always call this method after SapTransfer::Destroy.

SapAcquisition | SapAcquisition Class Members | Basic Class Hierarchy Chart

SapAcquisition::GetConfigFile, SapAcquisition::SetConfigFile

const char ***GetConfigFile**();

virtual BOOL **SetConfigFile**(const char **configFile*);

Remarks

Gets/sets the name of the acquisition configuration file (CCF).

You normally set the initial value for this attribute in the SapAcquisition constructor. If you use the default constructor, then this value is NULL.

You can only call SetConfigFile before the Create method.

SapAcquisition | SapAcquisition Class Members | Basic Class Hierarchy Chart

SapAcquisition::GetLabel

const char ***GetLabel**();

Remarks

Gets a text description of the acquisition resource. This attribute is initially set to an empty string. After a successful call to the Create method, it is composed of the name of the server where the acquisition resource is located and the name of the acquisition resource: *ServerName [ResourceName]*.

SapAcquisition | SapAcquisition Class Members | Basic Class Hierarchy Chart

SapAcquisition::GetEventType, SapAcquisition::SetEventType

SapAcquisition::EventType **GetEventType**();

virtual BOOL **SetEventType**(SapAcquisition::EventType *eventType*);

Remarks

Gets/sets the combination of registered acquisition event types. The initial value for this attribute is EventNone, unless you specify another value in the constructor.

You can only call SetEventType before the Create method. See the SapAcquisition constructor for possible values for *eventType* .

SapAcquisition | SapAcquisition Class Members | Basic Class Hierarchy Chart

SapAcquisition::SetCallbackInfo

virtual BOOL **SetCallbackInfo**(SapAcqCallback *pCallback*, void **pContext* = NULL);

Remarks

Sets the application callback method for acquisition events and the associated context.

You can only call SetCallbackInfo before the Create method. See the SapAcquisition constructor for more details.

SapAcquisition | SapAcquisition Class Members | Basic Class Hierarchy Chart

SapAcquisition::GetCallback

SapAcqCallback **GetCallback**();

Remarks

Gets the current application callback method for acquisition events. The initial value for this attribute is NULL, unless you specify another value in the constructor.

See the SapAcquisition constructor for more details.

SapAcquisition | SapAcquisition Class Members | Basic Class Hierarchy Chart

SapAcquisition::GetContext

void ***GetContext**();

Remarks

Gets the application context associated with acquisition events. The initial value for this attribute is NULL, unless you specify another value in the constructor.

See the SapAcquisition constructor for more details.

SapAcquisition | SapAcquisition Class Members | Basic Class Hierarchy Chart

SapAcquisition::GetCamSel, SapAcquisition::SetCamSel

int **GetCamsel**();

virtual BOOL **SetCamSel**(int *camSel*);

Remarks

Specifies the zero-based index of the camera input from which the acquisition device grabs images. The maximum value allowed depends on the acquisition hardware and the current data format.

The initial value for this attribute is 0. It is then set according to the current acquisition device value when calling the Create method.

You cannot call SetCamSel during live acquisition, that is, when the SapTransfer::IsGrabbing method returns TRUE.

SapAcquisition | SapAcquisition Class Members | Basic Class Hierarchy Chart

SapAcquisition::IsLutEnabled

BOOL **IsLutEnabled**();

Remarks

Gets the current LUT enable value. The initial value for this attribute is FALSE. It is then set according to the current the acquisition device value when calling the Create method.

SapAcquisition | SapAcquisition Class Members | Basic Class Hierarchy Chart

SapAcquisition::CanEnableLut

BOOL **CanEnableLut**();

Remarks

Checks if the acquisition lookup table may be enabled/disabled. The initial value for this attribute is FALSE. It is then set according to the current the acquisition device capability when calling the Create method.

SapAcquisition | SapAcquisition Class Members | Basic Class Hierarchy Chart

SapAcquisition::GetFlipMode, SapAcquisition::SetFlipMode

SapAcquisition::FlipMode **GetFlipMode**();

virtual BOOL **SetFlipMode**(SapAcquisition::FlipMode *flipMode*);

Parameters

flipMode	SapAcquisition::FlipNone	No flipping
	SapAcquisition::FlipHorizontal	Acquired images are flipped horizontally
	SapAcquisition::FlipVertical	Acquired images are flipped vertically

Remarks

Gets/sets the flipping (that is, mirroring) mode for acquired images. The initial value for this attribute is 'FlipNone'.

You can only call SetFlipMode after the Create method.

SapAcquisition | SapAcquisition Class Members | Basic Class Hierarchy Chart

SapAcquisition::GetNumPlanarInputs

int **GetNumPlanarInputs**();

Remarks

Gets the number of cameras used for acquiring into vertical planar buffers, where a value of 1 means that planar mode is disabled. All cameras must be synchronized together. The returned value is only meaningful after you call the Create method.

SapAcquisition | SapAcquisition Class Members | Basic Class Hierarchy Chart

SapAcquisition::GetPlanarInputs, SapAcquisition::SetPlanarInputs

BOOL **GetPlanarInputs**(BOOL **pCamEnable*);

BOOL **SetPlanarInputs**(BOOL **pCamEnable*, int *numCameras*);

Parameters

pCamEnable	Camera configuration array, must have at least 32 entries
numCameras	Number of cameras to configure for planar acquisition

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets/sets the current configuration for synchronous acquisition into vertical planar buffers, where all cameras are synchronized together.

Individual entries in the *pCamEnable* array are set to TRUE if the corresponding camera is enabled for planar acquisition; otherwise, they are set to FALSE. The entry at index 0 in *pCamEnable* corresponds to the first camera, the entry at index 1 corresponds to the second camera, and so on. If planar mode is disabled, then only the entry at index 0 is set.

You can only call GetPlanarInputs and SetPlanarInputs after the Create method.

SapAcquisition | SapAcquisition Class Members | Basic Class Hierarchy Chart

SapAcquisition::IsFlatFieldAvailable

BOOL **IsFlatFieldAvailable**();

Remarks

Gets availability of hardware-based flat-field correction. You can only call IsFlatFieldAvailable after the Create method.

SapAcquisition | SapAcquisition Class Members | Basic Class Hierarchy Chart

SapAcquisition::IsBayerAvailable

BOOL **IsBayerAvailable**();

Remarks

Gets availability of hardware-based Bayer conversion. You can only call IsBayerAvailable after the Create method.

SapAcquisition | SapAcquisition Class Members | Basic Class Hierarchy Chart

SapAcquisition::GetSerialPortName

BOOL **GetSerialPortName**(char **serialPortName*);

Parameters

serialPortName Memory area large enough to receive the text for the serial port name (at least 64 bytes)

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the name of the serial port attached to the current acquisition device.

You can only call GetSerialPortName after the Create method.

SapAcquisition | SapAcquisition Class Members | Basic Class Hierarchy Chart

SapAcquisition::SaveParameters

virtual BOOL **SaveParameters**(const char **configFile*);

Parameters

configFile Name of the acquisition configuration file (CCF) for saving camera and frame grabber related acquisition parameters

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Saves the current values of acquisition module parameters to the specified file.

SapAcquisition | SapAcquisition Class Members | Basic Class Hierarchy Chart

SapAcquisition::GetLut

SapLut ***GetLut**();

Remarks

Gets the current acquisition lookup table that has already been automatically created and initialized when calling the Create method. You may manipulate the LUT through the methods in the SapLut Class, and reprogram it using the ApplyLut method.

GetLut returns NULL if the current acquisition device does not support lookup tables.

SapAcquisition | SapAcquisition Class Members | Basic Class Hierarchy Chart

SapAcquisition::ApplyLut

virtual BOOL **ApplyLut**(BOOL *enable* = TRUE);

Parameters

enable Optionally enable the lookup table after reprogramming

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Reprograms the acquisition lookup table. After getting the current LUT using the GetLut method, use the methods in the SapLut Class to manipulate it. Then use ApplyLut to apply the changes. You need to enable the LUT in order to affect acquired images. Note that some acquisition devices do not support enabling or disabling the LUT.

SapAcquisition | SapAcquisition Class Members | Basic Class Hierarchy Chart

SapAcquisition::EnableLut

virtual BOOL **EnableLut**(BOOL *enable* = TRUE);

Parameters

enable TRUE to enable the acquisition lookup table, FALSE to disable it

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Enables or disables the acquisition lookup table. When the LUT is disabled, it does not affect acquired images. However, its contents are not lost, so they may be used again without reprogramming the acquisition hardware. Note that some acquisition devices do not support this feature.

SapAcquisition | SapAcquisition Class Members | Basic Class Hierarchy Chart

SapAcquisition::IsSignalStatusAvailable

BOOL **IsSignalStatusAvailable**();

BOOL **IsSignalStatusAvailable**(SapAcquisition::SignalStatus *signalStatus*);

Parameters

signalStatus Status signal to inquire. One or more of the following values may be ORed together.

SapAcquisition::SignalNone	No signal
SapAcquisition::SignalHSyncPresent	Horizontal sync signal (analog video source) or line valid (digital video source)
SapAcquisition::SignalVSyncPresent	Vertical sync signal (analog video source) or frame valid (digital video source)
SapAcquisition::SignalPixelClkPresent	Pixel clock signal
SapAcquisition::SignalChromaPresent	Color burst signal (valid for NTSC and PAL)
SapAcquisition::SignalHSyncLock	Successful lock to an horizontal sync signal, for an analog video source
SapAcquisition::SignalVSyncLock	Successful lock to a vertical sync signal, for an analog video source
SapAcquisition::SignalPowerPresent	Power is available for a camera. This does not necessarily mean that power is used by the camera, it only indicates that power is available at the camera connector, where it might be supplied from the board PCI bus or from the board PC power connector. The

returned value value is FALSE if the circuit fuse is blown, therefore power cannot be supplied to any connected camera.

Return Value

Returns TRUE if the acquisition device can detect the specified status signals, FALSE otherwise

Remarks

Reports the availability of the status of input signals connected to the acquisition device. Use the first form of IsSignalStatusAvailable to inquire about all input signals. Use the second form to narrow the inquiry down to specific signals only.

SapAcquisition | SapAcquisition Class Members | Basic Class Hierarchy Chart

SapAcquisition::GetSignalStatus

virtual BOOL **GetSignalStatus**(SapAcquisition::SignalStatus *signalStatus*, BOOL ***pIsActive**);

virtual BOOL **GetSignalStatus**(SapAcquisition::SignalStatus ***pSignalStatus**,
SapAcqCallback *pCallback* = NULL, void ***pContext** = NULL);

Parameters

signalStatus	Combination of status signals to inquire. See the SapAcquisition::IsSignalStatusAvailable method for a list of possible values.
pIsActive	Set upon return to TRUE if the specified status signals have been detected, FALSE otherwise
pSignalStatus	Set upon return to the combination of detected status signals. See the SapAcquisition::IsSignalStatusAvailable method for a list of possible values.
pCallback	Application callback function to be called each time the combination of detected signal status changes. The callback function must be declared as: void MyCallback(SapAcqCallbackInfo ***pInfo**);
pContext	Optional pointer to an application context to be passed to the callback function. If *pCallback* is NULL, this parameter is ignored.

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Reports the status of input signals connected to the acquisition device. Use the first form of GetSignalStatus for a one-time inquiry. Since many signals may be detected at the same time, values are usually combined together using a bitwise OR operation.

The second form allows asynchronous notification of application code whenever the combination of status signals changes. This may happen, for example, when an input cable is accidentally disconnected. First call the

method as follows:

GetSignalStatus(&currentStatus, MyCallback, &myContext);

This first reads the current value of the signal status. An internal mechanism then periodically checks for signal status changes, and notifies the application program using the callback function. You must call GetSignalStatus again with a NULL argument to disable the application callback function:

GetSignalStatus(&currentStatus, NULL);

SapAcquisition | SapAcquisition Class Members | Basic Class Hierarchy Chart

SapAcquisition::SoftwareTrigger

virtual BOOL **SoftwareTrigger**(SapAcquisition::SoftwareTriggerType *triggerType*);

Parameters

triggerType	Trigger type may be one of the following values	
	SapAcquisition::SoftwareTriggerExt	External trigger
	SapAcquisition::SoftwareTriggerExtFrame	External frame trigger
	SapAcquisition::SoftwareTriggerExtLine	External line trigger

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Simulates a trigger to the acquisition device. Use SoftwareTrigger for testing purposes when the actual hardware trigger is not available.

Note that in order for this feature to work, external trigger must be enabled. This can be done either through CamExpert or by calling the SetParameter method for the CORACQ_PRM_EXT_TRIGGER_ENABLE parameter.

Also, this feature may not be implemented on the current acquisition device. To find out if it is, call the GetCapability method for the CORACQ_CAP_SOFTWARE_TRIGGER capability.

SapAcquisition | SapAcquisition Class Members | Basic Class Hierarchy Chart

SapAcquisition::IsCapabilityValid

virtual BOOL **IsCapabilityValid**(int *cap*);

Parameters

cap Low-level Sapera C library capability to be checked

Return Value

Returns TRUE if the capability is supported, FALSE otherwise

Remarks

Checks for the availability of a low-level Sapera C library capability for the acquisition module. Call this method before GetCapability to avoid invalid or not available capability errors.

IsCapabilityValid is rarely needed. The SapAcquisition class already uses important capabilities internally for self-configuration and validation.

See the *Sapera LT Acquisition Parameters Reference Manual* for a description of all capabilities and their possible values.

SapAcquisition | SapAcquisition Class Members | Basic Class Hierarchy Chart

SapAcquisition::IsParameterValid

virtual BOOL **IsParameterValid**(int *param*);

Parameters

param Low-level Sapera C library parameter to be checked

Return Value

Returns TRUE if the parameter is supported, FALSE otherwise

Remarks

Checks for the availability of a low-level Sapera C library parameter for the acquisition module. Call this method before GetParameter to avoid invalid or not available parameter errors.

IsParameterValid is rarely needed. The SapAcquisition class already uses important parameters internally for self-configuration and validation.

See the *Sapera LT Acquisition Parameters Reference Manual* for a description of all parameters and their possible values.

SapAcquisition | SapAcquisition Class Members | Basic Class Hierarchy Chart

SapAcquisition::GetCapability

virtual BOOL **GetCapability**(int *cap*, void **pValue*);

Parameters

cap	Low-level Sapera C library capability to read
pValue	Pointer to capability value to read back

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

This method allows direct read access to low-level Sapera C library capabilities for the acquisition module. It needs a pointer to a memory area large enough to receive the capability value, which is usually a 32-bit integer.

GetCapability is rarely needed. The SapAcquisition class already uses important capabilities internally for self-configuration and validation.

See the *Sapera LT Acquisition Parameters Reference Manual* for a description of all capabilities and their possible values.

SapAcquisition | SapAcquisition Class Members | Basic Class Hierarchy Chart

SapAcquisition::GetParameter, SapAcquisition::SetParameter

virtual BOOL **GetParameter**(int *param*, void **pValue*);

virtual BOOL **SetParameter**(int *param*, int *value*, BOOL *updateNow* = TRUE);

virtual BOOL **SetParameter**(int *param*, void **pValue*, BOOL *updateNow* = TRUE);

Parameters

param	Low-level Sapera C library parameter to read or write
pValue	Pointer to parameter value to read back or to write
value	New parameter value to write
updateNow	Allows delayed updating of acquisition parameters

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

These methods allow direct read/write access to low-level Sapera C library parameters for the acquisition module. The GetParameter method needs a pointer to a memory area large enough to receive the parameter value that is usually a 32-bit integer. The first form of SetParameter accepts a 32-bit value for the new value. The second form takes a pointer to the new value, and is required when the parameter uses more than 32-bits of

storage.

By default, *updateNow* is TRUE, therefore calling SetParameter programs the acquisition hardware with the new value immediately. However, some parameters should not be set individually, as this may result in inconsistencies and error conditions in the acquisition resource.

If *updateNow* is FALSE, new parameter values are accumulated internally. The next time SetParameter is called with *updateNow* set to TRUE, all the new values are sent in one operation to the acquisition hardware, thus avoiding the problems just described.

Note that you will rarely need to use these methods. You should first make certain that what you need is not already supported by the SapAcquisition Class. Also, directly setting parameter values may interfere with the correct operation of the class.

See the *Sapera LT Acquisition Parameters Reference Manual* for a description of all parameters and their possible values.

SapAcquisition | SapAcquisition Class Members | Basic Class Hierarchy Chart

SapAcquisition::CustomCommand

virtual BOOL **CustomCommand**(int *command*, void **inData*, int *inDataSize*, void **outData*, int *outDataSize*);

Parameters

command	Low-level command ID
inData	Memory area with input data
inDataSize	Number of bytes of input data
outData	Memory area to receive output data
outDataSize	Maximum number of bytes of output data

Return Value

Returns TRUE if successful, FALSE otherwise

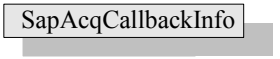
Remarks

Provides a way to directly call custom commands specific to the acquisition hardware.

You will rarely need to use this method since the functionality is usually customer or OEM specific.

SapAcquisition | SapAcquisition Class Members | Basic Class Hierarchy Chart

SapAcqCallbackInfo



The SapAcqCallbackInfo Class acts as a container for storing all arguments to the callback function for the SapAcquisition Class.

#include <SapClassBasic.h>

SapAcqCallbackInfo Class Members | Basic Class Hierarchy Chart

SapAcqCallbackInfo Class Members

Construction	
SapAcqCallbackInfo	Class constructor
Attributes	
GetAcquisition	Gets the SapAcquisition object associated with acquisition events or signal status reporting
GetContext	Gets the application context associated with acquisition events or signal status reporting
GetEventType	Gets the acquisition events that triggered the call to the application callback
GetEventCount	Gets the current count of acquisition events
GetSignalStatus	Gets the input signal status that triggered the call to the application callback

SapAcqCallbackInfo | Basic Class Hierarchy Chart

Member Functions

The following functions are members of the SapAcqCallbackInfo Class.

SapAcqCallbackInfo::SapAcqCallbackInfo

SapAcqCallbackInfo(
  SapAcquisition **pAcq*,
  void **pContext*,
  SapAcquisition::EventType *eventType*,
  int *eventCount*
);

SapAcqCallbackInfo(
  SapAcquisition **pAcq*,
  void **pContext*,
  SapAcquisition::SignalStatus *signalStatus*
);

Parameters

pAcq	SapAcquisition object that calls the callback function.
pContext	Pointer to the application context.
eventType	Combination of acquisition events. See the SapAcquisition constructor for a list a possible values.
eventCount	Current acquisition event count.
signalStatus	Combination of signal status values. See SapAcquisition::IsSignalStatusAvailable for a list a possible values.

Remarks

SapAcquisition objects create an instance of this class before each call to the acquisition callback method in order to combine all function arguments into one container.

SapAcquisition uses this class for two different purposes. The first case applies to reporting acquisition events. The *pContext* parameter takes the value specified in the SapAcquisition class constructor; *eventType* identifies the combination of events that triggered the call to the callback function; and *eventCount* increments by one at each call, starting at 1.

The second case applies to reporting signal status changes. The *pContext* parameter takes the value specified in the SapAcquisition::GetSignalStatus method, and *signalStatus* identifies the new signal status that triggered the call to the callback function.

SapAcqCallbackInfo | SapAcqCallbackInfo Class Members | Basic Class Hierarchy Chart

SapAcqCallbackInfo::GetAcquisition

SapAcquisition ***GetAcquisition**();

Remarks

Gets the SapAcquisition object context associated with acquisition events or signal status reporting. See the SapAcquisition constructor for more details.

SapAcqCallbackInfo | SapAcqCallbackInfo Class Members | Basic Class Hierarchy Chart

SapAcqCallbackInfo::GetContext

void ***GetContext**();

Remarks

Gets the application context associated with acquisition events or signal status reporting. See the SapAcquisition constructor and SapAcquisition::GetSignalStatus for more details.

SapAcqCallbackInfo | SapAcqCallbackInfo Class Members | Basic Class Hierarchy Chart

SapAcqCallbackInfo::GetEventType

SapAcquisition::EventType **GetEventType**();

Remarks

Gets the combination of acquisition events that triggered the call to the application callback. See the SapAcquisition constructor for the list of possible values.

SapAcqCallbackInfo | SapAcqCallbackInfo Class Members | Basic Class Hierarchy Chart

SapAcqCallbackInfo::GetEventCount

int **GetEventCount**();

Remarks

Gets the current count of acquisition events. The initial value is 1 and increments after every call to the acquisition callback function.

SapAcqCallbackInfo | SapAcqCallbackInfo Class Members | Basic Class Hierarchy Chart

SapAcqCallbackInfo::GetSignalStatus

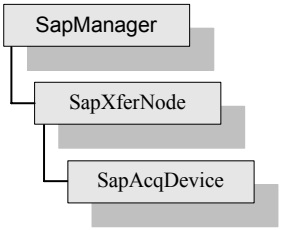
SapAcquisition::SignalStatus **GetSignalStatus**();

Remarks

Gets the input signal status that triggered the call to the application callback. See SapAcquisition::GetSignalStatus for the list of possible values.

SapAcqCallbackInfo | SapAcqCallbackInfo Class Members | Basic Class Hierarchy Chart

SapAcqDevice



The *SapAcqDevice* Class provides the functionality for reading/writing features from/to devices such as a GigE-Vision camera. The class also contains functions for sending commands and registering events to devices.

This class is used as a source transfer node to allow data transfers from an acquisition device to another transfer node, such as a buffer, for example.

Note: Frame-grabber devices are not supported by this class. The *SapAcquisition* class must be used in such cases.#include <SapClassBasic.h>

SapAcqDevice Class Members | Basic Class Hierarchy Chart

SapAcqDevice Class Members

Construction	
SapAcqDevice	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
General Parameters	
GetConfigFile SetConfigFile	Gets/sets the name of the acquisition configuration file (CCF)
GetReadOnly SetReadOnly	Gets/sets whether or not the class has read-only access to the device
GetUpdateFeatureMode SetUpdateFeatureMode	Gets/sets the mode by which features are written to the device
GetLabel	Gets a text description of the acquisition device
GetConfigName SetConfigName	Gets/sets the configuration name to be used when saving the device features using the *SaveFeatures* method
GetModeName SetModeName	Gets/sets the mode name to be used when saving the device features using the *SaveFeatures* method
Feature Access	
GetFeatureCount	Returns the number of features supported by the acquisition device
GetFeatureNameByIndex	Returns the name of a feature associated with a specified index
GetFeatureIndexByName	Returns the index of a feature associated with a specified name
IsFeatureAvailable	Returns whether or not a feature is supported by the acquisition device
GetFeatureInfo	Returns information on a feature associated with a specified name or index
GetFeatureValue	Returns the value of a feature associated with a specified name or index
SetFeatureValue	Sets the value of a feature associated with a specified name or index
UpdateFeaturesFromDevice	Gets all the features from the acquisition device at once
UpdateFeaturesToDevice	Sets all the features to the acquisition device at once
LoadFeatures	Loads all the features from a configuration file
SaveFeatures	Saves all (or a subset of) features to a configuration file.
Event Management	
GetEventCount	Returns the number of events supported by the acquisition device
GetEventNameByIndex	Returns the name of an event associated with a specified index

GetEventIndexByName	Returns the index of an event associated with a specified name
IsEventAvailable	Returns whether or not an event is supported by the acquisition device
RegisterCallback	Registers a callback function for the event associated with a specified name or index
UnregisterCallback	Unregisters a callback function on the event associated with a specified name or index
IsCallbackRegistered	Returns whether or not a callback function was registered on the event associated with a specified name or index

SapAcqDevice | Basic Class Hierarchy Chart

SapAcqDevice::SapAcqDevice

SapAcqDevice(SapLocation *location* = SapLocation::ServerSystem, BOOL *readOnly* = FALSE);
SapAcqDevice(SapLocation *location*, const char **configFile*);

Parameters

location	SapLocation object specifying the server where the acquisition device is located and the index of the acquisition device on this server.
readOnly	TRUE to force read-only access to the device. If another application is already accessing the device (through this class) use this function to obtain read-only access to the device. To know what functions of the *SapAcqDevice* class are accessible with this option, refer to the function documentation.
configFile	Name of the acquisition configuration file (CCF) that describes all the acquisition parameters. A CCF file can be created using the *CamExpert* utility.

Remarks

The *SapAcqDevice* constructor does not actually create the low-level Sapera resources. To do this, you must call the SapAcqDevice::Create method.

The first constructor is used when no configuration file is required. In such a case the default parameters of the acquisition device are used. You can optionally specify a read-only option to obtain a read-only access to the device. This option is useful only when another application has already obtained a read-write access to the same device.

The second constructor allows you to load a configuration file (CCF) previously created by the CamExpert tool or by your own application.

The *SapAcqDevice* object is used only for storing the acquisition device parameters. To acquire data, use the SapTransfer Class (or one of its derived classes) and pass the *SapAcqDevice* object as a parameter for the constructor. *SapTransfer* then handles the actual data transfer.

SapAcqDevice | SapAcqDevice Class Members | Basic Class Hierarchy Chart

SapAcqDevice::Create

virtual BOOL **Create**();

Remarks

Creates all the low-level Sapera resources needed by the acquisition object. Always call this method before SapTransfer::Create.

Return Value

Returns TRUE if the object was successfully created, FALSE otherwise

SapAcqDevice | SapAcqDevice Class Members | Basic Class Hierarchy Chart

SapAcqDevice::Destroy

virtual BOOL **Destroy**();

Remarks

Destroys all the low-level Sapera resources needed by the acquisition object. Always call this method after SapTransfer::Destroy.

Return Value

Returns TRUE if the object was successfully destroyed, FALSE otherwise

SapAcqDevice | SapAcqDevice Class Members | Basic Class Hierarchy Chart

SapAcqDevice::GetConfigFile, SapAcqDevice::SetConfigFile

const char ***GetConfigFile**() const;
BOOL **SetConfigFile**(const char **configFile*);

Parameters

configFile Name of the configuration file (CCF) to be loaded at creation

Remarks

Gets/sets the name of the acquisition configuration file (CCF) to be loaded at creation, that is, when the Create method is called.

You normally set the initial value for this attribute in the *SapAcqDevice* constructor. If you use the default constructor, then this value is NULL.

You can only call *SetConfigFile* before the SapAcqDevice::Create method.

SapAcqDevice | SapAcqDevice Class Members | Basic Class Hierarchy Chart

SapAcqDevice::GetConfigName, SapAcqDevice::SetConfigName

const char ***GetConfigName**() const;
BOOL **SetConfigName**(const char **configName*);

Parameters

configName Name of the configuration to be written to the CCF file. The length of the string must not exceed 64 characters.

Remarks

Gets/sets the configuration name to be used when saving the device features using the *SaveFeatures* method. It is then possible to uniquely identify different configuration files when the company name, camera model name, and mode name are the same. For example, “High Contrast” might be used as configuration name.

When loading a configuration file using *LoadFeatures*, this parameter is automatically updated.

SapAcqDevice | SapAcqDevice Class Members | Basic Class Hierarchy Chart

SapAcqDevice::GetEventCount

BOOL **GetEventCount**(int **eventCount*);

Parameters

eventCount Number of events supported by the acquisition device

Remarks

Returns the number of events supported by the acquisition device. Devices do not necessarily support the same event set. For instance you can use this function to retrieve the number of events and then get the name of those event using *GetEventNameByIndex*.

Return Value

Returns TRUE if the function was successful, FALSE otherwise

SapAcqDevice | SapAcqDevice Class Members | Basic Class Hierarchy Chart

SapAcqDevice::GetEventIndexByName

BOOL **GetEventIndexByName**(const char **eventName*, int **eventIndex*);

Parameters

eventName	Event name. See device User's Manual for the list of supported events.
eventIndex	Returns the index of the event associated with the specified name

Remarks

Returns the index of an event associated with a specified name.

Return Value

Returns TRUE if the function was successful, FALSE otherwise

SapAcqDevice | SapAcqDevice Class Members | Basic Class Hierarchy Chart

SapAcqDevice::GetEventNameByIndex

BOOL **GetEventNameByIndex**(int *eventIndex*, char **eventName*, int *eventNameSize*);

Parameters

eventIndex	Index of the event. Ranges from 0 to *GetEventCount*.
eventName	Returns the name of the event associated with the specified index
eventNameSize	Size (in bytes) of the buffer pointed to by *eventName*

Remarks

Returns the name of an event associated with a specified index. The typical usage is converting an event index (retrieved from your callback information) to the corresponding name. You can then sort the events by name in order to call the appropriate event handling code.

Return Value

Returns TRUE if the function was successful, FALSE otherwise

SapAcqDevice | SapAcqDevice Class Members | Basic Class Hierarchy Chart

SapAcqDevice::GetFeatureCount

BOOL **GetFeatureCount**(int **featureCount*);

Parameters

featureCount Number of features supported by the acquisition device

Remarks

Returns the number of features supported by the acquisition device. Devices do not necessarily support the same feature set. For instance you can use this function to retrieve the number of features and then get information about those features using *GetFeatureInfo*.

Return Value

Returns TRUE if the function was successful, FALSE otherwise

SapAcqDevice | SapAcqDevice Class Members | Basic Class Hierarchy Chart

SapAcqDevice::GetFeatureIndexByName

BOOL **GetFeatureIndexByName**(const char **featureName*, int **featureIndex*);

Parameters

featureName Name of the feature. See device User's Manual for the list of supported features.

featureIndex Returns the index of the feature associated with the specified name

Remarks

Returns the index of a feature associated with a specified name. This function is useful in building a list of indexes associated with the feature names you commonly use. Then those features are accessed by index to increase performance.

Return Value

Returns TRUE if the function was successful, FALSE otherwise

SapAcqDevice | SapAcqDevice Class Members | Basic Class Hierarchy Chart

SapAcqDevice::GetFeatureInfo

BOOL **GetFeatureInfo**(const char **featureName*, SapFeature **feature*);
BOOL **GetFeatureInfo**(int *featureIndex*, SapFeature **feature*);

Parameters

featureName	Name of the feature. See device User's Manual for the list of supported features.
featureIndex	Index of the feature. Ranges from 0 to *GetFeatureCount*.
feature	Pointer to a *SapFeature* object to store feature information

Remarks

Returns information on a feature associated with a specified name or index. All information about the feature is stored in a *SapFeature* object. The *SapFeature* object contains the attributes of the feature such as name, type, range, etc. See the SapFeature class for more details.

The *SapFeature* object must be contructed and created before passing it to this function.

Return Value

Returns TRUE if the function was successful, FALSE otherwise

SapAcqDevice | SapAcqDevice Class Members | Basic Class Hierarchy Chart

SapAcqDevice::GetFeatureNameByIndex

BOOL **GetFeatureNameByIndex**(int *featureIndex*, char **featureName*, int *featureNameSize*);

Parameters

featureIndex	Index of the feature. Ranges from 0 to *GetFeatureCount*.
featureName	Returns the name of the feature associated with the specified index
featureNameSize	Size (in bytes) of the buffer pointed to by *featureName*

Remarks

Returns the name of a feature associated with a specified index. For instance you can use this function to display the name of all features supported by the device.

Return Value

Returns TRUE if the function was successful, FALSE otherwise

SapAcqDevice | SapAcqDevice Class Members | Basic Class Hierarchy Chart

SapAcqDevice::GetFeatureValue

BOOL **GetFeatureValue**(const char **featureName*, UINT32 **featureValue*);
BOOL **GetFeatureValue**(const char **featureName*, UINT64 **featureValue*);
BOOL **GetFeatureValue**(const char **featureName*, float **featureValue*);
BOOL **GetFeatureValue**(const char **featureName*, double **featureValue*);
BOOL **GetFeatureValue**(const char **featureName*, BOOL **featureValue*);
BOOL **GetFeatureValue**(const char **featureName*, char **featureString*, int *featureStringSize*);
BOOL **GetFeatureValue**(const char **featureName*, SapBuffer **featureBuffer*);
BOOL **GetFeatureValue**(const char **featureName*, SapLut **featureLut*);

BOOL **GetFeatureValue**(int *featureIndex*, UINT32 **featureValue*);
BOOL **GetFeatureValue**(int *featureIndex*, UINT64 **featureValue*);
BOOL **GetFeatureValue**(int *featureIndex*, float **featureValue*);
BOOL **GetFeatureValue**(int *featureIndex*, double **featureValue*);
BOOL **GetFeatureValue**(int *featureIndex*, BOOL **featureValue*);
BOOL **GetFeatureValue**(int *featureIndex*, char **featureString*, int *featureStringSize*);
BOOL **GetFeatureValue**(int *featureIndex*, SapBuffer **featureBuffer*);
BOOL **GetFeatureValue**(int *featureIndex*, SapLut **featureLut*);

Parameters

featureName	Name of the feature. See device User's Manual for the list of supported features.
featureIndex	Index of the feature. Ranges from 0 to *GetFeatureCount*.
featureValue	Returns the value of the specified feature. You must choose the overloaded function according to the feature type
featureString	Returns the content of a string feature
featureStringSize	Size (in bytes) of the buffer pointed to by *featureString*
featureBuffer	Address of a SapBuffer object (already constructed and created) to retrieve a buffer feature
featureLut	Address of a SapBuffer object (already constructed and created) to retrieve a LUT feature

Remarks

Returns the value of a feature associated with a specified name of index.

To determine which overloaded function to use you must obtain the type of the feature using *GetFeatureInfo*. In the case of a class type (such as *SapBuffer* or *SapLut*) the object must be constructed and created before calling this function.

Return Value

Returns TRUE if the function was successful, FALSE otherwise

Note

Except for unitless features, each feature has its specific native unit, for example millisecond, KHz, tenth of degree, etc. This information is obtained through the *SapFeature::GetSiUnit* and *SapFeature::GetSiToNativeExp10* functions.

SapAcqDevice | SapAcqDevice Class Members | Basic Class Hierarchy Chart

SapAcqDevice::GetLabel

const char ***GetLabel**() const;

Remarks

Gets a text description of the acquisition device. This attribute is initially set to an empty string. After a successful call to the *Create* method, it is composed of the name of the server where the acquisition device is located and the name of the acquisition device: *ServerName [DeviceName]*.

SapAcqDevice | SapAcqDevice Class Members | Basic Class Hierarchy Chart

SapAcqDevice::GetModeName, SapAcqDevice::SetModeName

const char ***GetModeName**() const;
BOOL **SetModeName**(const char **modeName*);

Parameters

modeName	Name of the camera mode to be written to the CCF file. The length of the string must not exceed 64 characters.

Remarks

Gets/sets the mode name to be used when saving the device features using the *SaveFeatures* method. It is then possible to uniquely identify different modes when the company name and camera model name are the same. For example, “Single-Channel, Free-Running” might be used as mode name.

When loading a configuration file using *LoadFeatures*, this parameter is automatically updated.

SapAcqDevice | SapAcqDevice Class Members | Basic Class Hierarchy Chart

SapAcqDevice::GetReadOnly, SapAcqDevice::SetReadOnly

BOOL **GetReadOnly**() const;
BOOL **SetReadOnly**(BOOL *readOnly*);

Parameters

readOnly TRUE to force read-only access to the device

Remarks

Gets/sets whether or not the class has read-only access to the device. See class contructor for more detail on this option. You can only call *SetReadOnly* before the *Create* method.

SapAcqDevice | SapAcqDevice Class Members | Basic Class Hierarchy Chart

SapAcqDevice::GetUpdateFeatureMode, SapAcqDevice::SetUpdateFeatureMode

UpdateFeatureMode **GetUpdateFeatureMode**() const;
BOOL **SetUpdateFeatureMode**(UpdateFeatureMode *mode*);

Parameters

mode

SapAcqDevice::UpdateFeatureAuto
Each feature is written to the device individually without being cached.

SapAcqDevice::UpdateFeatureManual
Each feature is temporarily cached until *UpdateFeaturesToDevice* is called.

Remarks

Gets/sets the mode by which features are written to the device. In the automatic mode (*SapAcqDevice::UpdateFeatureAuto*) every time a feature is set using the *SetFeatureValue* function the feature value is immediately written to the device. In the manual mode (*SapAcqDevice::UpdateFeatureAuto*) each feature is temporarily cached until the SapAcqDevice::UpdateFeaturesToDevice is called to write all features to the device at once.

SapAcqDevice | SapAcqDevice Class Members | Basic Class Hierarchy Chart

SapAcqDevice::IsCallbackRegistered

BOOL **IsCallbackRegistered**(const char **eventName*, BOOL **isRegistered*);
BOOL **IsCallbackRegistered**(int *eventIndex*, BOOL **isRegistered*);

Parameters

eventName	Name of the event. See device User's Manual for the list of supported events.
eventIndex	Index of the event. Ranges from 0 to *GetEventCount*.
isRegistered	TRUE if a callback function was registered on this event. FALSE otherwise

Remarks

Returns whether or not a callback function was registered on the event associated with a specified name or index. For example use this function in a loop to determine if the callback function associated with the current index has to be unregistered.

Return Value

Returns TRUE if the function was successful, FALSE otherwise

SapAcqDevice | SapAcqDevice Class Members | Basic Class Hierarchy Chart

SapAcqDevice::IsEventAvailable

BOOL **IsEventAvailable**(const char **eventName*, BOOL **isAvailable*);

Parameters

eventName	Name of the event. See device User's Manual for the list of supported events.
isAvailable	TRUE if the event is supported by the device. FALSE otherwise

Remarks

Returns whether or not an event is supported by the acquisition device. This function is useful when an application supports several acquisition devices each having a different event set.

Return Value

Returns TRUE if the function was successful, FALSE otherwise

SapAcqDevice | SapAcqDevice Class Members | Basic Class Hierarchy Chart

SapAcqDevice::IsFeatureAvailable

BOOL **IsFeatureAvailable**(const char **featureName*, BOOL ***isAvailable*);

Parameters

featureName	Name of the feature. See device User's Manual for the list of supported features.
isAvailable	TRUE if the feature is supported by the device. FALSE otherwise

Remarks

Returns whether or not a feature is supported by the acquisition device. This function is useful when an application supports several acquisition devices each having a different feature set.

Return Value

Returns TRUE if the function was successful, FALSE otherwise

SapAcqDevice | SapAcqDevice Class Members | Basic Class Hierarchy Chart

SapAcqDevice::LoadFeatures

BOOL **LoadFeatures**(const char **configFile*);

Parameters

configFile	Name of the configuration file (CCF) to load the feature from

Remarks

Loads all the features from a configuration file. This function reads the features from a *Sapera LT's CCF File* and writes them to the device. The CCF file is generated by the *Sapera LT CamExpert Program*.

For devices that support hardware persistence storage (see your Acquisition Device User's Manual) loading a CCF file is not mandatory. For other devices you must load a CCF file to ensure the device is in a usable state.

Return Value

Returns TRUE if the function was successful, FALSE otherwise

Note

This function is not valid if the object was contructed with read-only access.

SapAcqDevice | SapAcqDevice Class Members | Basic Class Hierarchy Chart

SapAcqDevice::RegisterCallback

BOOL **RegisterCallback**(const char **eventName*, SapAcqDeviceCallback *callback*, void **context*);
BOOL **RegisterCallback**(int *eventIndex*, SapAcqDeviceCallback *callback*, void **context*);

Parameters

eventName	Name of the event. See device User's Manual for the list of supported events.
eventIndex	Index of the event. Ranges from 0 to *GetEventCount*.

callback	Address of a user callback function of the following form:
	`void MyCallback(SapAcqDeviceCallbackInfo* pInfo)`
	`{`
	`}`

context	Pointer to a user storage (that is, variable, structure, buffer, etc). Can be NULL.

Remarks

Registers an event by associating a callback function to the specified name or index. When the event occurs in the acquisition device the specified callback function is called. The callback function provides information on the corresponding event (in the *SapAqcDeviceCallbackInfo* object). Refer to the *SapAqcDeviceCallbackInfo* class for more detail on the available information. The context pointer is also returned by the callback function allowing you to exchange user information between the callback and your application context.

Return Value

Returns TRUE if the function was successful, FALSE otherwise

Note

This function is not valid if the object was contructed with read-only access.

Example

```
void MyCallback(SapAcqDeviceCallbackInfo *pInfo)
{
   // Access information using functions of SapAcqDeviceCallbackInfo class
   // ...
}

main()
{
   // ...
   acqDevice.RegisterCallback("FeatureValueChanged", MyCallback, NULL);
   // ...
   acqDevice.UnregisterCallback("FeatureValueChanged");
   // ...
}
```

SapAcqDevice | SapAcqDevice Class Members | Basic Class Hierarchy Chart

SapAcqDevice::SaveFeatures

BOOL **SaveFeatures**(const char **configFile*);

Parameters

configFile Name of the configuration file (CCF) to save the features to

Remarks

Saves all (or a subset of) the device features to a configuration file. This function reads the features from the device and writes them to a *Sapera LT's CCF File*.

Not all features are saved. For example read-only features are not saved by default. To know what features are saved by default refer to *SapFeature::IsSavedToConfigFile* function. Also the *SapFeature::SetSavedToConfigFile* function also allows you to control whether each individual feature is saved or not.

This function is useful for devices that do not support hardware persistence storage (see the Acquisition Device User's Manual) in order to retreive the feature values after a shut down of the device.

Return Value

Returns TRUE if the function was successful, FALSE otherwise

SapAcqDevice | SapAcqDevice Class Members | Basic Class Hierarchy Chart

SapAcqDevice::SetFeatureValue

BOOL **SetFeatureValue**(const char **featureName*, UINT32 *featureValue*);
BOOL **SetFeatureValue**(const char **featureName*, UINT64 *featureValue*);
BOOL **SetFeatureValue**(const char **featureName*, float *featureValue*);
BOOL **SetFeatureValue**(const char **featureName*, double *featureValue*);
BOOL **SetFeatureValue**(const char **featureName*, BOOL *featureValue*);
BOOL **SetFeatureValue**(const char **featureName*, const char **featureString*);
BOOL **SetFeatureValue**(const char **featureName*, SapBuffer **featureBuffer*);
BOOL **SetFeatureValue**(const char **featureName*, SapLut **featureLut*);

BOOL **SetFeatureValue**(int *featureIndex*, UINT32 *featureValue*);
BOOL **SetFeatureValue**(int *featureIndex*, UINT64 *featureValue*);
BOOL **SetFeatureValue**(int *featureIndex*, float *featureValue*);
BOOL **SetFeatureValue**(int *featureIndex*, double *featureValue*);
BOOL **SetFeatureValue**(int *featureIndex*, BOOL *featureValue*);
BOOL **SetFeatureValue**(int *featureIndex*, const char **featureString*);
BOOL **SetFeatureValue**(int *featureIndex*, SapBuffer **featureBuffer*);
BOOL **SetFeatureValue**(int *featureIndex*, SapLut **featureLut*);

Parameters

featureName	Name of the feature. See device User's Manual for the list of supported features.
featureIndex	Index of the feature. Ranges from 0 to *GetFeatureCount*.
featureValue	Value of the feature to write to device. You must choose the overloaded function according to the feature type
featureString	String to write to device
featureBuffer	Address of a SapBuffer object (already constructed and created) to write to device
featureLut	Address of a SapBuffer object (already constructed and created) to write to device

Remarks

Writes the value of a feature associated with a specified name of index to the device.

To determine which overloaded function to use you must obtain the type of the feature using *GetFeatureInfo*. In the case of a class type (such as *SapBuffer* or *SapLut*) the object must be constructed and created before calling this function.

Return Value

Returns TRUE if the function was successful, FALSE otherwise

Note

Except for unitless features, each feature has its specific native unit, for example millisecond, KHz, tenth of degree, etc. This information is obtained through the *SapFeature::GetSiUnit* and *SapFeature::GetSiToNativeExp10* functions.

This function is not valid if the object was contructed with read-only access.

SapAcqDevice | SapAcqDevice Class Members | Basic Class Hierarchy Chart

SapAcqDevice::UnregisterCallback

BOOL **UnregisterCallback**(const char **eventName*);
BOOL **UnregisterCallback**(int *eventIndex*);

Parameters

eventName	Name of the event. See device User's Manual for the list of supported events.
eventIndex	Index of the event. Ranges from 0 to *GetEventCount*.

Remarks

Unregisters a callback function on the event associated with a specified name or index. Use this function in a loop to unregister all the callback functions previously registered.

Return Value

Returns TRUE if the function was successful, FALSE otherwise

Note

This function is not valid if the object was contructed with read-only access.

Example

```
// Unregisters all the callback functions
//
UINT32 eventCount, eventIndex;
acqDevice.GetEventCount(&eventCount);
for (eventIndex = 0; eventIndex < eventCount; eventIndex++)
{
   BOOL isRegistered;
   acqDevice.IsCallbackRegistered(eventIndex, &isRegistered);
   if (isRegistered)
   {
      acqDevice.UnregisterCallback(eventIndex);
   }
}
```

SapAcqDevice | SapAcqDevice Class Members | Basic Class Hierarchy Chart

SapAcqDevice::UpdateFeaturesFromDevice

BOOL **UpdateFeaturesFromDevice**();

Remarks

Gets all the features from the acquisition device at once. This function must be used when the *feature update mode* is set to manual (see SapAcqDevice::GetUpdateFeatureMode, SapAcqDevice::SetUpdateFeatureMode). In this mode writing individual features is done to an internal cache. Calling this function resets the internal cache to the values currently present in the device. This is useful when a certain number of features have been set to the internal cache but you want to undo those settings.

Return Value

Returns TRUE if the function was successful, FALSE otherwise

Note

This function is not valid if the object was contructed with read-only access.

SapAcqDevice | SapAcqDevice Class Members | Basic Class Hierarchy Chart

SapAcqDevice::UpdateFeaturesToDevice

BOOL **UpdateFeaturesToDevice**();

Remarks

Sets all the features to the acquisition device at once. This function must be used when the *feature update mode* is set to manual (see e SapAcqDevice::GetUpdateFeatureMode, SapAcqDevice::SetUpdateFeatureMode). In this mode writing individual features is done to an internal cache. After all the required features have been set, call this function to write them all to the device.

Return Value

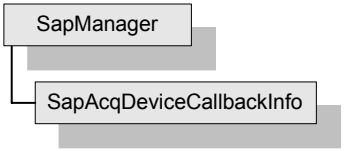
Returns TRUE if the function was successful, FALSE otherwise

Note

This function is not valid if the object was contructed with read-only access.

SapAcqDevice | SapAcqDevice Class Members | Basic Class Hierarchy Chart

SapAcqDeviceCallbackInfo



The *SapAcqDeviceCallbackInfo* class acts as a container for storing all arguments to the callback function for the SapAcqDevice class.

#include <SapClassBasic.h>

SapAcqDeviceCallbackInfo Class Members | Basic Class Hierarchy Chart

SapAcqDeviceCallbackInfo Class Members

Attributes

GetAcqDevice	Gets the *SapAcqDevice* object associated with acquisition events
GetContext	Gets the application context associated with acquisition events

GetEventInfo	Gets the low-level Sapera handle of the event info resource
Common Parameters	
GetEventCount	Returns the number of events that have occurred since the callback function was registered
GetEventIndex	Returns the index of the event
GetHostTimeStamp	Returns the time stamp corresponding to the moment when the event occurred on the host
GetAuxiliaryTimeStamp	Returns the time stamp corresponding to the moment when the event occurred on the device
Custom Parameters	
GetCustomData	Returns the data associated with a custom event
GetCustomSize	Returns the size of the custom data returned by *GetCustomData*
Unnamed Parameters	
GetGenericParam0..3	Returns generic parameters shared by the different events
Event-specific Parameters	
GetFeatureIndex	Returns the index of the feature associated with the event

SapAcqDeviceCallbackInfo | Basic Class Hierarchy Chart

SapAcqDeviceCallbackInfo::GetAcqDevice

SapAcqDevice ***GetAcqDevice**() const;

Remarks

Gets the *SapAcqDevice* object associated with acquisition events. See the *SapAcqDevice*constructor for more details.

SapAcqDeviceCallbackInfo | SapAcqDeviceCallbackInfo Class Members | Basic Class Hierarchy Chart

SapAcqDeviceCallbackInfo::GetAuxiliaryTimeStamp

BOOL **GetAuxiliaryTimeStamp**(UINT64 **auxTimeStamp*);

Parameters

auxTimeStamp Address of a 64-bit integer to return the time stamp value

Remarks

Returns the time stamp corresponding to the moment when the event occurred on the device. Not all the devices support this time stamp. This value is specific to the device. See the device driver User's Manual for more information on the availability of this value and the associated unit.

SapAcqDeviceCallbackInfo | SapAcqDeviceCallbackInfo Class Members | Basic Class Hierarchy Chart

SapAcqDeviceCallbackInfo::GetContext

void ***GetContext**() const;

Remarks

Gets the application context associated with acquisition events. See the *SapAcqDevice::RegisterCallback* function for more details.

SapAcqDeviceCallbackInfo | SapAcqDeviceCallbackInfo Class Members | Basic Class Hierarchy Chart

SapAcqDeviceCallbackInfo::GetCustomData

BOOL **GetCustomData**(void ***customData*);

Parameters

customData Address of a pointer to receive the address to the data buffer

Remarks

Returns the address of a buffer containing the data associated with a custom event. You must not free the buffer after you are finished using it.

This parameter is supported only in special versions of certain device drivers. See the device driver User's Manual for more information on the availability of this parameter.

Example

```
void MyCallback(SapAcqDeviceCallbackInfo *pInfo)
{
   // Retrieve the data buffer
   void* pCustomData;
   pInfo->GetCustomData(&pCustomData);

   // Use the data buffer
   //...
}
```

SapAcqDeviceCallbackInfo | SapAcqDeviceCallbackInfo Class Members | Basic Class Hierarchy Chart

SapAcqDeviceCallbackInfo::GetCustomSize

BOOL **GetCustomSize**(int **customSize*);

Parameters

customSize Address of an integer to return the value

Remarks

Returns the size of the custom data returned by *GetCustomData*.

SapAcqDeviceCallbackInfo | SapAcqDeviceCallbackInfo Class Members | Basic Class Hierarchy Chart

SapAcqDeviceCallbackInfo::GetEventCount

BOOL **GetEventCount**(int **eventCount*);

Parameters

eventCount Address of an integer to return the count

Remarks

Returns the number of events that have occurred since the callback function was registered. This value is a positive integer starting at 1.

SapAcqDeviceCallbackInfo | SapAcqDeviceCallbackInfo Class Members | Basic Class Hierarchy Chart

SapAcqDeviceCallbackInfo::GetEventIndex

BOOL **GetEventIndex**(int **eventIndex*);

Parameters

eventIndex Address of an integer to return the index

Remarks

Returns the index of the current event. Use this index to retrieve the name of the event using the *SapAcqDevice::GetEventNameByIndex* function.

SapAcqDeviceCallbackInfo | SapAcqDeviceCallbackInfo Class Members | Basic Class Hierarchy Chart

SapAcqDeviceCallbackInfo::GetEventInfo

COREVENTINFO **GetEventInfo**() const;

Remarks

Gets the low-level Sapera handle of the event info resource. This function should not be used unless a handle to the low-level C API is required to access some functionality not exposed in the C++ API.

SapAcqDeviceCallbackInfo | SapAcqDeviceCallbackInfo Class Members | Basic Class Hierarchy Chart

SapAcqDeviceCallbackInfo::GetFeatureIndex

BOOL **GetFeatureIndex**(int **featureIndex*);

Parameters

featureIndex Address of an integer to return the index

Remarks

Returns the index of the feature associated with the event. For example it is used by the "*FeatureInfoChanged*" and "*FeatureValueChanged*" events of the *SapAcqDevice* class. In this case it represents the index of the feature whose attributes or value have changed. This index ranges from 0 to *SapAcqDevice::GetFeatureCount* – 1.

SapAcqDeviceCallbackInfo | SapAcqDeviceCallbackInfo Class Members | Basic Class Hierarchy Chart

SapAcqDeviceCallbackInfo::GetGenericParam0..3

BOOL **GetGenericParam0**(int **paramValue*);
BOOL **GetGenericParam1**(int **paramValue*);
BOOL **GetGenericParam2**(int **paramValue*);
BOOL **GetGenericParam3**(int **paramValue*);

Parameters

paramValue Address of an integer to return the value

Remarks

Returns the four generic parameters shared by the different events. You should use aliases instead when they are available. Otherwise refer to the device driver User's Manual for a list of events using generic parameters.

SapAcqDeviceCallbackInfo | SapAcqDeviceCallbackInfo Class Members | Basic Class Hierarchy Chart

SapAcqDeviceCallbackInfo::GetHostTimeStamp

BOOL **GetHostTimeStamp**(UINT64 **hostTimeStamp*);

Parameters

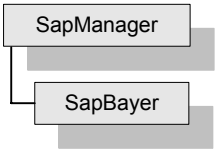
hostTimeStamp Address of a 64-bit integer to return the time stamp value

Remarks

Returns the time stamp corresponding to the moment when the event occurred on the host. Under Windows the value corresponding to the high-resolution performance counter is directly returned. Refer to *QueryPerformanceCounter* in the Windows API documenation for more detail on how to convert this value to time units

SapAcqDeviceCallbackInfo | SapAcqDeviceCallbackInfo Class Members | Basic Class Hierarchy Chart

SapBayer



The purpose of the SapBayer Class is to support conversion of Bayer encoded images. In the first case, images are acquired from a Bayer camera. They are then converted to RGB either by the acquisition device (if supported) or through software. In the second case, images are taken from another source (for example, loaded from disk). Only the software implementation is then available

#include <SapClassBasic.h>

SapBayer Class Members | Base Class: SapManager | Basic Class Hierarchy Chart

SapBayer Class Members

Construction	
SapBayer	Class constructor
Create	Allocates the internal resources
Destroy	Releases the internal resources
Attributes	
GetAcquisition, SetAcquisition	Gets/sets the acquisition object for acquiring Bayer images
GetBuffer, SetBuffer	Gets/sets the buffer object in which images are acquired or loaded
GetBayerBuffer	Gets the buffer object used as the destination for software conversion
GetBayerBufferCount SetBayerBufferCount	Gets/sets the number of buffer resources used for software conversion
IsEnabled	Checks if Bayer conversion is enabled
IsSoftware	Checks if Bayer conversion is performed in software or using the hardware

GetAlign SetAlign	Gets/sets the Bayer alignment mode
GetAvailAlign	Gets the available alignment modes
GetMethod SetMethod	Gets/sets the pixel value calculation method
GetAvailMethod	Gets the available pixel value calculation methods
GetWBGain SetWBGain	Gets/sets the white balance gain coefficients
GetWBOffset SetWBOffset	Gets/sets the white balance offset coefficients
GetGamma SetGamma	Gets/sets the gamma correction factor for the Bayer lookup table
GetOutputFormat SetOutputFormat	Gets/sets the data output format of Bayer conversion
IsLutEnabled	Gets the current Bayer lookup table enable value
IsAcqLut	Checks if the Bayer lookup table corresponds to the acquisition LUT
Operations	
Enable	Enables/disables Bayer conversion
Convert	Converts a Bayer-encoded image to an RGB image using software
WhiteBalance	Calculates the white balance gain coefficients for Bayer conversion
GetLut	Gets the current Bayer lookup table
EnableLut	Enables/disables the Bayer lookup table

SapBayer | Base Class: SapManager | Basic Class Hierarchy Chart

Member Functions

The following functions are members of the SapBayer Class.

SapBayer::SapBayer

SapBayer(
  SapAcquisition **pAcq,*
  SapBuffer **pBuffer*
);

SapBayer(
  SapBuffer **pBuffer*
);

Parameters

pAcq	SapAcquisition object to use for image acquisition and Bayer conversion (if available in hardware)
pBuffer	SapBuffer object in which images will be acquired or loaded

Remarks

The SapBayer constructor does not actually create the internal resources. To do this, you must call the Create method.

When using hardware conversion, the result will be stored in the buffer object identified by *pBuffer*.

When using software conversion, the buffer object for the result of the conversion is automatically created using relevant attributes from *pBuffer*.

In both cases, the resulting SapBuffer object will be available through the GetBayerBuffer method.

SapBayer | SapBayer Class Members | Basic Class Hierarchy Chart

SapBayer::Create

virtual BOOL **Create**();

Return Value

Returns TRUE if the object was successfully created, FALSE otherwise

Remarks

Creates all the internal resources needed by the Bayer conversion object.

If the Bayer object is associated with a SapAcquisition object (using the SapBayer constructor or the SetAcquisition method), then you can only call this method after the Create method for the acquisition object.

If there is no acquisition object, then you can only call this method after the Create method for the associated buffer object instead (specified using the SapBayer constructor or the SetBuffer method).

SapBayer | SapBayer Class Members | Basic Class Hierarchy Chart

SapBayer::Destroy

virtual BOOL **Destroy**();

Return Value

Returns TRUE if the object was successfully destroyed, FALSE otherwise

Remarks

Destroys all the internal resources needed by the Bayer conversion object

SapBayer | SapBayer Class Members | Basic Class Hierarchy Chart

SapBayer::GetAcquisition, SapBayer::SetAcquisition

SapAcquisition ***GetAcquisition**();

virtual BOOL **SetAcquisition**(SapAcquisition **pAcq*);

Remarks

Gets/sets the SapAcquisition object to be used for image acquisition and for Bayer conversion. You can only call SetAcquisition before the Create method.

SapBayer | SapBayer Class Members | Basic Class Hierarchy Chart

SapBayer::GetBuffer, SapBayer::SetBuffer

SapBuffer ***GetBuffer**();

virtual BOOL **SetBuffer**(SapBuffer **pBuffer*);

Remarks

Gets/sets the SapBuffer object in which images will be acquired or loaded.

For software conversion, the buffer format must be either SapFormatMono8 or SapFormatMono16. The buffer object with the result of the conversion is then available by calling the GetBayerBuffer method.

For hardware conversion, the buffer format may be SapFormatRGB888, SapFormatRGB8888, or SapFormatRGB101010 (16-bit input image only). In this case, the buffer object returned by this method is the same as the one returned by calling the GetBayerBuffer method.

You can only call SetBuffer before the Create method.

SapBayer | SapBayer Class Members | Basic Class Hierarchy Chart

SapBayer::GetBayerBuffer

SapBuffer ***GetBayerBuffer**();

Remarks

Gets the buffer object used as the destination for software conversion. When using software conversion, this object is automatically created using relevant attributes from the main buffer object (the one in which images are acquired or loaded).

When Bayer conversion is performed in hardware, this method returns the same buffer object as the GetBuffer method.

SapBayer | SapBayer Class Members | Basic Class Hierarchy Chart

SapBayer::GetBayerBufferCount, SapBayer::SetBayerBufferCount

int **GetBayerBufferCount**();

virtual BOOL **SetBayerBufferCount**(int *bayerBufferCount*);

Parameters

bayerBufferCount Number of buffer resources

Remarks

Gets/sets the number of buffer resources used for software conversion. The initial value for this attribute is 2.

You can only call SetBayerBufferCount before the Create method.

SapBayer | SapBayer Class Members | Basic Class Hierarchy Chart

SapBayer::IsEnabled

BOOL **IsEnabled**();

Remarks

Checks if Bayer conversion is enabled. The initial value for this attribute depends on the acquisition device.

Use the Enable method if you need to enable or disable Bayer conversion.

SapBayer | SapBayer Class Members | Basic Class Hierarchy Chart

SapBayer::IsSoftware

BOOL **IsSoftware**();

Remarks

Returns TRUE if Bayer conversion is not available in the acquisition device, or if software conversion has been explicitly chosen by calling the Enable method.

Returns FALSE if Bayer conversion is available in the acquisition device, and software conversion has not been explicitly chosen by calling the Enable method.

SapBayer | SapBayer Class Members | Basic Class Hierarchy Chart

SapBayer::GetAlign, SapBayer::SetAlign

SapBayer::Align **GetAlign**();

virtual BOOL **SetAlign**(SapBayer::Align *align*);

Parameters

align Bayer alignment mode may be one of the following values

SapBayer::AlignGBRG

SapBayer::AlignBGGR

SapBayer::AlignRGGB

SapBayer::AlignGRBG

Remarks

Gets/sets the Bayer alignment mode, which must correspond to the upper left 2x2 square of the Bayer scheme of the camera.

The initial value for this attribute is SapBayer::AlignGRBG. It is then set to the acquisition device Bayer alignment value when calling the Create method (except when no acquisition device is used).

SapBayer | SapBayer Class Members | Basic Class Hierarchy Chart

SapBayer::GetAvailAlign

SapBayer::Align **GetAvailAlign**();

Remarks

Gets the valid Bayer alignment modes, combined together using bitwise OR.

The initial value for this attribute includes all available modes. It is then set to the valid acquisition device alignment modes when calling the Create method (except when no acquisition device is used).

See the GetAlign method for a list of possible alignment modes.

SapBayer | SapBayer Class Members | Basic Class Hierarchy Chart

SapBayer::GetMethod, SapBayer::SetMethod

SapBayer::Method **GetMethod**();

virtual BOOL **SetMethod**(SapBayer::Method *method*);

Parameters

method	Bayer pixel value calculation method may be one of the following values	
	SapBayer::Method1	Technique based on bilinear interpolation. Fast, but tends to smooth the edges of the image.
	SapBayer::Method2	Advanced technique, better for preserving the edges of the image. However, it works well only when the image has a strong content in green. Otherwise, little amounts of noise may be visible within objects.
	SapBayer::Method3	Advanced technique, almost as good as Method2 for preserving the edges, but independent of the image content in green. Small colour artefacts of 1 pixel may be visible at the edges.

Remarks

Gets/sets the Bayer pixel value calculation method.

The initial value for this attribute is SapBayer::Method1. It is then set to the acquisition device Bayer method when calling the Create method (except when no acquisition device is used).

SapBayer | SapBayer Class Members | Basic Class Hierarchy Chart

SapBayer::GetAvailMethod

SapBayer::Method **GetAvailMethod**();

Remarks

Gets the valid Bayer pixel value calculation methods, combined together using bitwise OR.

The initial value for this attribute includes all available methods. It is then set to the valid acquisition device calculation methods when calling the Create method (except when no acquisition device is used).

See the GetMethod method for a list of possible calculation methods.

SapBayer | SapBayer Class Members | Basic Class Hierarchy Chart

SapBayer::GetWBGain, SapBayer::SetWBGain

SapDataFRGB **GetWBGain**();

virtual BOOL **SetWBGain**(SapDataFRGB *wbGain*);

Parameters

wbGain New white balance gain coefficients

Remarks

Gets/sets the Bayer white balance gain coefficients. These may also be calculated automatically using the WhiteBalance method.

The white balance gain coefficients are the red, green, and blue gains applied to the input image before filtering. These are used to balance the three color components so that a pure white at the input gives a pure white at the output. Set all gains to 1.0 if no white balance gain is required.

The initial value for this attribute is 1.0 for each color component.

SapBayer | SapBayer Class Members | Basic Class Hierarchy Chart

SapBayer::GetWBOffset, SapBayer::SetWBOffset

SapDataFRGB **GetWBOffset**();

virtual BOOL **SetWBOffset**(SapDataFRGB *wbOffset*);

Parameters

wbOffset New white balance offset coefficients

Remarks

Gets/sets the Bayer white balance offset coefficients. These apply only for hardware conversion, that is, when the IsSoftware method returns FALSE.

The white balance offset coefficients are the red, green, and blue offsets applied to the input image before filtering. These are used to balance the three color components so that a pure white at the input gives a pure white at the output. Set all offsets to 0.0 if no white balance offset is required.

The initial value for this attribute is 0.0 for each color component.

SapBayer | SapBayer Class Members | Basic Class Hierarchy Chart

SapBayer::GetGamma, SapBayer::SetGamma

float **GetGamma**();

virtual BOOL **SetGamma**(float *gamma*);

Parameters

gamma New gamma correction factor

Remarks

Gets/sets the Bayer gamma correction factor. If Bayer conversion is enabled, and the Bayer lookup table is also enabled (using the EnableLut method), then Gamma correction with the specified *factor* is applied after Bayer conversion has been performed.

The initial value for this attribute is 1.0, which effectively disables Gamma correction.

SapBayer | SapBayer Class Members | Basic Class Hierarchy Chart

SapBayer::GetOutputFormat, SapBayer::SetOutputFormat

SapFormat **GetOutputFormat**();

virtual BOOL **SetOutputFormat** (SapFormat *format*);

Parameters

format New Bayer conversion output format

Remarks

Gets/sets the data output format of Bayer conversion. The only two possible values for this attribute are SapFormatRGB8888 and SapFormatRGB101010.

The initial value for this attribute is SapFormatUnknown. It is then set to the appropriate value when calling the Create method, or through SetOutputFormat. You can only call SetOutputFormat before the Create method

SapBayer | SapBayer Class Members | Basic Class Hierarchy Chart

SapBayer::IsLutEnabled

BOOL **IsLutEnabled**();

Remarks

Gets the current Bayer lookup table enable value. When enabled, this LUT is applied to image data after Bayer conversion has been performed.

The initial value for this attribute is FALSE. Use the EnableLut method to enable or disable the lookup table.

SapBayer | SapBayer Class Members | Basic Class Hierarchy Chart

SapBayer::IsAcqLut

BOOL **IsAcqLut** ();

Remarks

Checks if the Bayer lookup table corresponds to the acquisition LUT. If the return value is FALSE, then a software lookup table is used instead.

The initial value for this attribute is FALSE. It is then set according to the current acquisition device lookup table availability when calling the Create method.

SapBayer | SapBayer Class Members | Basic Class Hierarchy Chart

SapBayer::Enable

virtual BOOL **Enable**(BOOL *enable* = TRUE, BOOL *useHardware* = TRUE);

Parameters

enable	TRUE to enable Bayer conversion, FALSE to disable it
useHardware	TRUE to use hardware conversion, FALSE to use the software implementation

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Enables/disables conversion of Bayer images to RGB. If you set *useHardware* to TRUE, and hardware conversion is not available, then this method returns FALSE. If you set *useHardware* to FALSE, then you must call the Convert method to perform the actual conversion.

Use the SapAcquisition::IsBayerAvailable method to determine if hardware correction is available in the acquisition device.

SapBayer | SapBayer Class Members | Basic Class Hierarchy Chart

SapBayer::Convert

virtual BOOL **Convert**();

virtual BOOL **Convert**(int *srcIndex*);

virtual BOOL **Convert**(int *srcIndex*, int *dstIndex*);

Parameters

srcIndex	Source buffer resource index
dstIndex	Destination buffer resource index

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

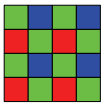
Converts a Bayer-encoded image to an RGB image using software.

The source buffer for the conversion is the current buffer resource in the main buffer object, unless you specify a source index. The GetBuffer method allows you to access this buffer.

The destination buffer for the conversion is the current buffer resource in the internal Bayer buffer object, unless you specify a destination index. The GetBayerBuffer method allows you to access this buffer.

The Bayer format assigns each pixel in a monochrome image the value of one color channel. RGB images are created by using neighboring pixel values to get the two missing color channels at each pixel.

Pixels in one row of a Bayer image alternate between the green channel value and either the red or the blue channel value. The default scheme is shown below.



The missing color channel values are determined using neighboring pixel values for the color channel in question by various methods, some of which are more computationally expensive, but give better image quality when the input image contains many strong edges.

SapBayer | SapBayer Class Members | Basic Class Hierarchy Chart

SapBayer::WhiteBalance

virtual BOOL **WhiteBalance**(int *x*, int *y*, int *width*, int *height*);

virtual BOOL **WhiteBalance**(SapBuffer **pBuffer*, int *x*, int *y*, int *width*, int *height*);

Parameters

x	Left coordinate of white balance region of interest
y	Top coordinate of white balance region of interest
width	Width of white balance region of interest
height	Height of white balance region of interest
pBuffer	Buffer object with the white balance region of interest

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Calculates the white balance gain coefficients needed for Bayer conversion. The region of interest of a Bayer-encoded image containing a uniformly illuminated white region. The intensity of the pixels should be as high as possible but not saturated. The coefficients are calculated as follows:

$$G_R = \mathrm{Max}(\overline{R}, \overline{G}, \overline{B}) / \overline{R}$$

$$G_G = \mathrm{Max}(\overline{R}, \overline{G}, \overline{B}) / \overline{G}$$

$$G_B = \mathrm{Max}(\overline{R}, \overline{G}, \overline{B}) / \overline{B}$$

where $\overline{R}$, $\overline{G}$ and $\overline{B}$ are the average values of each color component calculated on all the pixels of the input image.

The buffer format must be either SapFormatMono8 or SapFormatMono16. The buffer resource at the current index in the main buffer object (the one in which images are acquired or loaded) is used, unless you explicitly specify another buffer object using the *pBuffer* argument..

SapBayer | SapBayer Class Members | Basic Class Hierarchy Chart

SapBayer::GetLut

SapLut ***GetLut**();

Remarks

Gets the current Bayer lookup table that is applied to image data after Bayer conversion has been performed, if the lookup table has been enabled using the EnableLut method.

For hardware conversion, this is actually the acquisition lookup table, which you may also obtain through the SapAcquisition::GetLut method. If the acquisition hardware has no lookup table, then the return value is NULL.

For software conversion, the lookup table is created automatically inside the SapBayer object so that it is compatible with the buffer object on which Bayer conversion is performed.

SapBayer | SapBayer Class Members | Basic Class Hierarchy Chart

SapBayer::EnableLut

virtual BOOL **EnableLut**(BOOL *enable* = TRUE);

Parameters

enable TRUE to enable the Bayer lookup table, FALSE to disable it

Return Value

Returns TRUE if successful, FALSE otherwise

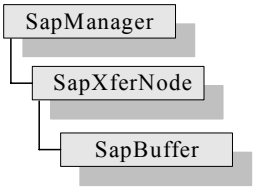
Remarks

Enables or disables the Bayer lookup table that is applied to image data after Bayer conversion has been performed.

For hardware conversion, this is actually the acquisition lookup table. For software conversion, the lookup table is created automatically inside the SapBayer object so that it is compatible with the buffer object on which Bayer conversion is performed.

SapBayer | SapBayer Class Members | Basic Class Hierarchy Chart

SapBuffer



The SapBuffer Class includes the functionality to manipulate an array of buffer resources. The array contains buffer resources with the same dimensions, format, and type.

The buffer object can be used as a destination transfer node to allow transferring data from a source node (such as acquisition, DSP, Pixel Processor, CAB, or another buffer) to a buffer resource. It can also be used as a source transfer node to allow transferring data from a buffer resource to another buffer or CAB resource. The array of buffers allows a transfer to cycle throughout all the buffers.

The buffer object can be displayed using the SapView Class and processed using the SapProcessing Class.

#include <SapClassBasic.h>

SapBuffer Class Members | Base Class: SapManager | Basic Class Hierarchy Chart

SapBuffer Class Members

Construction	
SapBuffer	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Attributes	
GetCount SetCount	Gets/sets the number of buffer resources in the array
GetWidth SetWidth	Gets/sets the width (in pixels) of all the buffer resources
GetHeight SetHeight	Gets/sets the height (in lines) of all the buffer resources

GetFormat SetFormat	Gets/sets the data format of all the buffer resources
GetType SetType	Gets/sets the type of all the buffer resources
SetVirtualAddress	Sets the virtual addresses to use for creating buffer resources
SetPhysicalAddress	Sets the physical addresses to use for creating buffer resources
GetParameters SetParameters	Gets/sets the count, width, height, format, and type of all the buffer resources
SetParametersFromFile	Sets the count, width, height, format, and type of all the buffer resources from an existing file storing a Sapera buffer
GetPixelDepth SetPixelDepth	Gets/sets the number of significant bits of all the buffer resources
GetFrameRate SetFrameRate	Gets/sets the frame rate of all the buffer resources
GetBytesPerPixel	Gets the number of bytes required to store a single buffer element
GetPitch	Gets the number of bytes between two consecutive lines of all the buffer resources
GetBufName	Gets the name of a buffer object that is shared between multiple processes
GetHandles	Gets the array of low-level Sapera handles to all the buffer resources
GetHandle operator[]	Gets the low-level Sapera handle of a specified buffer resource
GetIndex SetIndex	Gets/sets the index of the current buffer
Operations	
ResetIndex	Initializes the current buffer index
Next	Increments the current buffer index
Clear	Clears the content of all the buffers
Load	Loads an image file into the current buffer resource
Save	Saves the current buffer resource to an image file
Copy	Copies contents of a single buffer resource from another SapBuffer object
CopyAll	Copies contents of all the buffer resources from another SapBuffer object
SplitComponents	Splits a color buffer into its individual monochrome components
MergeComponents	Merges individual monochrome components into a color buffer
Read	Reads a consecutive series of pixel values in the current buffer resource
ReadElement	Reads the pixel value at a specified position in the current buffer resource

ReadLine	Reads a series of linearly positioned pixel values in the current buffer resource
ReadRect	Reads a series of pixel values from a rectangular area in the current buffer resource
Write	Writes a consecutive series of pixel values in the current buffer resource
WriteElement	Writes the pixel value at a specified position in the current buffer resource
WriteLine	Writes a series of linearly positioned pixel values to the current buffer resource
WriteRect	Writes a series of pixel values to a rectangular area in the current buffer resource
GetState	Gets the empty/full state of the current buffer resource
SetState	Sets the empty/full state of the current buffer resource
SetAllState	Gets the empty/full state of all the buffer resources
GetPage	Gets the active page of the current buffer resource for planar buffer types
SetPage	Sets the active page of the current buffer resource for planar buffer types
SetAllPage	Sets the active page of all the buffer resources for planar buffer types
GetAddress	Initiates direct address to buffer resource data by a pointer
ReleaseAddress	End direct buffer resource data access
IsMapped	Indicates if there currently exists a valid virtual data address for a buffer resource
GetCounterStamp	Gets a unique identifier associated with a buffer resource
GetSpaceUsed	Gets the number of data bytes actually stored in a buffer resource
Register	Allows remote access to all buffer resources
Unregister	Removes remote access capability for all buffer resources
GetCapability	Gets the value of a low-level Sapera C library capability
GetParameter SetParameter	Gets/sets the value of a low-level Sapera C library parameter

SapBuffer | Base Class: SapManager | Basic Class Hierarchy Chart

Member Functions

The following functions are members of the SapBuffer Class.

SapBuffer::SapBuffer

```
SapBuffer(
   int count = SapDefBufferCount,
   int width = SapDefWidth,
   int height = SapDefHeight,
   SapFormat format = SapDefFormat,
   SapBuffer::Type type = SapDefBufferType,
   SapLocation loc = SapLocation::ServerSystem
);
SapBuffer(
   int count,
   int physAddress[]
   int width = SapDefWidth,
   int height = SapDefHeight,
   SapFormat format = SapDefFormat,
   SapBuffer::Type type = SapBuffer::TypeContiguous,
);
SapBuffer(
   int count,
   void *virtAddress[]
   int width = SapDefWidth,
   int height = SapDefHeight,
   SapFormat format = SapDefFormat,
   SapBuffer::Type type = SapBuffer::TypeScatterGather,
);
SapBuffer(
   int count,
   SapXferNode *pSrcNode,
   SapBuffer::Type type = SapDefBufferType,
   SapLocation loc =  SapLocation::ServerSystem
);
SapBuffer(
   const char *fileName,
   SapBuffer::Type type = SapDefBufferType
   SapLocation loc =  SapLocation::ServerSystem
```

);

SapBuffer(
  int *count*,
  const char **bufName*
  int *width* = SapDefWidth,
  int *height* = SapDefHeight,
  SapFormat *format* = SapDefFormat,
  SapBuffer::Type *type* = SapBuffer::TypeScatterGather,
  SapLocation *loc* = SapLocation::ServerSystem
);

SapBuffer(
  int *count*,
  const char **bufName*
  SapXferNode **pSrcNode*,
  SapBuffer::Type *type* = SapDefBufferType,
  SapLocation *loc* = SapLocation::ServerSystem
);

SapBuffer(
  const char **bufName*
  int *startIndex*
  int *count*,
  SapBuffer::Type *type* = SapBuffer::TypeVirtual,
  SapLocation *loc* = SapLocation::ServerSystem
);

Parameters

count	Number of buffer resources
width	Width (in pixels) of all the buffer resources
height	Height (in lines) of all the buffer resources
format	Data format of all the buffer resources, can be one of the following values:

Monochrome (unsigned)	
SapFormatMono1	1-bit
SapFormatMono8	8-bit
SapFormatMono16	16-bit
SapFormatMono32	32-bit
Monochrome (signed)	
SapFormatInt8	8-bit
SapFormatInt16	16-bit
SapFormatInt32	32-bit

RGB Color

SapFormatRGB5551	16-bit (5 for each of red/green/blue, 1for alpha)
SapFormatRGB565	16-bit (5 for red, 6 for green, 5 for blue)
SapFormatRGB888	24-bit (8 for red, 8 for green, 8 for blue)
SapFormatRGB8888	32-bit (8 for each of red/green/blue, 8 for alpha)
SapFormatRGB101010	32-bit (10 for each of red/green/blue, 2 unused)
SapFormatRGB161616	48-bit (16 for each of red/green/blue)
SapFormatRGB16161616	64-bit (16 for each of red/green/blue/alpha)
SapFormatRGBP8	8-bit planar
SapFormatRGBP16	16-bit planar

YUV Color

SapFormatUYVY	16-bit, 4:2:2 subsampled
SapFormatYUY2	16-bit, 4:2:2 subsampled
SapFormatYVYU	16-bit, 4:2:2 subsampled
SapFormatYUYV	16-bit, 4:2:2 subsampled
SapFormatY411	12-bit, 4:1:1 subsampled
SapFormatY211	8-bit, 4:2:2 subsampled
SapFormatYUV	32-bit (8 for each of Y/U/V, 8 for alpha)

Other Formats

SapFormatHSV	32-bit HSV (8 for each component, 8 unused)
SapFormatHSI	32-bit HSI (8 for each component, 8 unused)
SapFormatHSIP8	8-bit HSI planar
SapFormatFloat	32-bit signed floating point
SapFormatPoint	64-bit (32-bit signed integer for both X and Y components)
SapFormatFPoint	64-bit (32-bit signed floating-point for both X and Y components)

See also the following wrapper classes for Sapera data elements: SapDataMono Class, SapDataRGB Class, SapDataRGBA Class, SapDataYUV Class, SapDataHSI Class, SapDataHSV Class, SapDataFloat Class, SapDataPoint Class, SapDataFPoint Class, and SapDataFRGB Class

type Type of all buffer resources can be one of the following values:

SapBuffer:: TypeContiguous
Buffers are allocated in Sapera Contiguous Memory, which is one large chunk of non-pageable and non-moveable memory reserved by Sapera at boot time. Buffer data is thus contained in a single memory block (not segmented). These buffers may be used as source and destination for transfer resources.

SapBuffer::TypeScatterGather
Buffers are allocated in noncontiguous memory (paged pool). Pages are locked in physical memory so that a scatter-gather list may be built. This allows allocation of very large buffers to be used as source and destination for transfer resources. The maximum amount of memory that may be allocated depends on available memory, the operating system, and the

application(s) used. If the amount of system memory exceeds 4 GBytes, Sapera automatically uses TypeScatterGatherUnmapped instead.

SapBuffer::TypeVirtual
Similar to TypeScatterGather, except that the memory pages are not locked. This allows allocation of very large buffers, but they cannot be used as source or destination for transfer resources.

SapBuffer::TypeOffscreen
Buffers are allocated in system memory. SapView objects created using these buffers may use display adapter hardware to copy from the buffer to video memory. System memory offscreen buffers may be created using any pixel format, but calling the SapView::Show method will take longer to execute if the display hardware does not efficiently support its pixel format.

SapBuffer::TypeOffscreenVideo
Buffers are allocated in offscreen video memory. SapView objects created using these buffers use display adapter hardware to perform a fast copy in video memory. These buffers are typically used when a graphical element is reused for several consecutive frames without modification. In this case, it is more efficient to keep this element in video memory and use display hardware capabilities.

SapBuffer::TypeOverlay
Buffers are allocated in video memory. Once you create SapView objects using these buffers and call their Show method one time, the display adapter overlay hardware will keep updating the display with the buffer contents with no additional calls The pixel format of overlay buffers must be supported by the display hardware. Typically, overlay buffers support more pixel formats (like YUV) than offscreen buffers. Also, color keying is supported for overlays. The SapView Class determines the behavior of the overlay regarding key colors.

SapBuffer::TypeDummy
Dummy buffers do not have any data memory. They may be used as placeholders by transfer resources when there is no physical data transfer.

SapBuffer::TypeUnmapped
Buffers are allocated as a series of non-contiguous chunks of physical memory. You may not access their data until they have been mapped to virtual memory addresses using the GetAddress method. This type of buffer is useful if the total amount of needed buffer data exceeds the amount of available virtual memory addresses (2 GBytes under Windows). To avoid a shortage of virtual memory addresses, use the ReleaseAddress method as soon as you are done accessing their data. Note that you cannot acquire images into these buffers.

SapBuffer::TypeScatterGatherUnmapped
These buffers are similar to TypeUnmapped, except that you can acquire images into them.

loc SapLocation object specifying the server on which the buffer resources are to be created. The resource index of the location object is ignored.

physAddress Array of physical addresses to use when creating buffer resources. This is intended for cases when you do not want Sapera to allocate buffer memory (in the Create method), and you

	already know the physical addresses where you want buffers to be located. These addresses typically correspond to hardware devices in the system.
virtAddress	Array of virtual addresses to use when creating buffer resources. This is intended for cases when you do not want Sapera to allocate buffer memory (in the Create method), but you want to control the allocation and free memory in the application program instead. Memory thus remains available even after calling the Destroy method.
pSrcNode	Source node object. The *width*, *height,* and *format* parameters are extracted automatically from this object. To ensure transfer compatibility, this object must match the source node specified when adding a transfer pair (SapXferPair) to the SapTransfer object.
fileName	Name of a Sapera image file from which to extract the *count, width*, *height,* and *format* parameters
bufName	Name identifying the buffer object so that it may be shared between multiple processes
startIndex	Starting index of buffer resource when using a shared buffer object created in another process

Remarks

The SapBuffer constructor does not actually create the low-level Sapera resources. To do this, you must call the Create method.

The *count* parameter specifies the number of buffer resources, all of which have the same *width*, *height*, *format*, and *type*.

Constructing the object using *physAddress* or *virtAddress* tells Sapera not to perform memory allocation itself in the Create method, but rather to rely on the supplied addresses.

Constructing the object using *pSrcNode* allows Sapera to automatically extract the *width*, *height*, and *format* from the source node to ensure transfer compatibility.

Constructing the object using *fileName* allows Sapera to automatically extract the *count*, *width*, *height*, and *format* from the file to ensure buffer compatibility. You must then use the Load method after calling Create.

The *loc* argument allows the creation of buffer resources on a remote server.

Constructing the object using *bufName* allows sharing of a buffer object between multiple processes. The first process that calls the constructor creates the actual buffer resources. The other processes that call the constructor with the same name automatically use the same resources. You may use the *startIndex* and *count* arguments to use only a subset of all the shared resources in the buffer object.

To transfer data to/from the buffer object, you must use the SapTransfer Class (or one of its derived classes) and specify the SapBuffer object as a parameter. The data transfer is then controlled by the SapTransfer Class.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::Create

virtual BOOL **Create**();

Return Value

Returns TRUE if the object was successfully created, FALSE otherwise

Remarks

Creates all the low-level Sapera resources needed by the buffer object. If it is used together with an acquisition and a transfer object, then you must call this method after SapAcquisition::Create, but before SapTransfer::Create.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::Destroy

virtual BOOL **Destroy**();

Return Value

Returns TRUE if the object was successfully destroyed, FALSE otherwise

Remarks

Destroys all the low-level Sapera resources needed by the buffer object. Always call this method after SapTransfer::Destroy.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::GetCount, SapBuffer::SetCount

int **GetCount**();

virtual BOOL **SetCount**(int *count*);

Remarks

Gets/sets the number of buffer resources. The initial value for this attribute is 1, unless you specify another value in the constructor.

You can only call SetCount before the Create method.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::GetWidth, SapBuffer::SetWidth

int **GetWidth**();

virtual BOOL **SetWidth**(int *width*);

Remarks

Gets/sets the width of all the buffer resources.

There are many possible initial values for this attribute, if you do not specify it explicitly in the constructor.

If using the constructor with a SapXferNode object, then this value is 0, and is then set correctly from the transfer node object after calling the Create method.

If using the constructor with a file name, then this value is taken directly from the file.

If using the constructor with a shared buffer with a starting index and count, then this value is 0. It is then set correctly from the shared buffer object after calling the Create method.

Otherwise, the initial value is equal to 640.

You can only call SetWidth before the Create method.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::GetHeight, SapBuffer::SetHeight

int **GetHeight**();

virtual BOOL **SetHeight**(int *height*);

Remarks

Gets/sets the height of all the buffer resources.

There are many possible initial values for this attribute, if you do not specify it explicitly in the constructor.

If using the constructor with a SapXferNode object, then this value is 0, and is then set correctly from the transfer node object after calling the Create method.

If using the constructor with a file name, then this value is taken directly from the file.

If using the constructor with a shared buffer object with a starting index and count, then this value is 0. It is then set correctly from the shared buffer object after calling the Create method.

Otherwise, the initial value is equal to 480.

You can only call SetHeight before the Create method.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::GetFormat, SapBuffer::SetFormat

SapFormat **GetFormat**();

virtual BOOL **SetFormat**(SapFormat *format*);

Remarks

Gets/sets the format of all the buffer resources.

There are many possible initial values for this attribute, if you do not specify it explicitly in the constructor.

If using the constructor with a SapXferNode object, then this value is SapFormatUnknown, and is then set correctly from the transfer node object after calling the Create method.

If using the constructor with a file name, then this value is taken directly from the file.

If using the constructor with a shared buffer object with a starting index and count, then this value is SapFormatUnknown. It is then set correctly from the shared buffer object after calling the Create method.

Otherwise, the initial value is equal to SapFormatMono8.

You can only call SetFormat before the Create method. See the SapBuffer constructor for possible values for *format* (other than SapFormatUnknown).

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::GetType, SapBuffer::SetType

SapBuffer::Type **GetType**();

virtual BOOL **SetType**(SapBuffer::Type *type*);

Remarks

Gets/sets the type of all the buffer resources.

There are many possible initial values for this attribute, if you do not specify it explicitly in the constructor.

If using the constructor with physical addresses, then this value is TypeContiguous.

If using the constructor with virtual addresses, then this value is TypeScatterGather.

If using the constructor with a shared buffer object with width/height/format, then this value is also TypeScatterGather.

If using the constructor with a shared buffer object with a starting index and count, then this value is TypeVirtual.

Otherwise, the initial value is equal to TypeDefault. It is then set to a valid value (almost always TypeScatterGather) after calling the Create method

You can only call SetType before the Create method. See the SapBuffer constructor for possible values for *type*.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::SetVirtualAddress

virtual BOOL **SetVirtualAddress**(void **virtAddress[]*);

Remarks

Sets the virtual addresses to use for creating buffer resources.

You can only call SetVirtualAddress before the Create method. See the SapBuffer constructor for more details.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::SetPhysicalAddress

virtual BOOL **SetPhysicalAddress**(int **physAddress[]*);

Remarks

Sets the physical addresses to use for creating buffer resources.

You can only call SetPhysicalAddress before the Create method. See the SapBuffer constructor for more details.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::GetParameters, SapBuffer::SetParameters

void **GetParameters**(int **count*, int **width*, int **height*, SapFormat **format*, SapBuffer::Type **type*);

virtual BOOL **SetParameters**(int *count*, int *width*, int *height*, SapFormat *format*, SapBuffer::Type *type*);

virtual BOOL **SetParameters**(int *count*, void **virtAddress[]*, int *width*, int *height*, SapFormat *format*, SapBuffer::Type *type*);

virtual BOOL **SetParameters**(int *count*, int **physAddress[]*, int *width*, int *height*, SapFormat *format*, SapBuffer::Type *type*);

Remarks

Gets/sets the count, width, height, format, type of all the buffer resources. You can also set the virtual and physical addresses to use when creating buffer resources.

You can only call SetParameters before the Create method. See the SapBuffer constructor for possible values for more details.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::SetParametersFromFile

virtual BOOL **SetParametersFromFile**(const char **fileName*, SapBuffer::Type *type*);

Remarks

Sets the count, width, height, format, and type of all the buffer resources from an existing Sapera image file to ensure buffer compatibility.

You can only call SetParametersFromFile before the Create method. You can then use the Load method after calling Create.

See the SapBuffer constructor for possible values for *type*.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::GetPixelDepth, SapBuffer::SetPixelDepth

int **GetPixelDepth**();

virtual void **SetPixelDepth**(int *pixelDepth*);

Remarks

Gets/sets the number of significant bits of all the buffer resources. The range of possible values is given by the SapManager::GetPixelDepthMin, SapManager::GetPixelDepthMax method.

The value of this attribute is only relevant after calling the Create method, during which it is set in one of the following ways, depending on which SapBuffer constructor was used.

If using a constructor with a SapXferNode object, the value is set from the pixel depth of this object.

Otherwise, the value is set according to the current buffer data format.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::GetFrameRate, SapBuffer::SetFrameRate

float **GetFrameRate**();

virtual void **SetFrameRate**(float *frameRate*);

Remarks

Gets/sets the frame rate in the buffer object. This value is used when loading or saving a sequence of buffers from/to a file (for example in AVI format).

When loading a buffer sequence the frame rate is restored from the file and can then be obtained through a call to *GetFrameRate*.

When saving a buffer sequence you may optionally save the frame rate. To do so you must specify the frame rate using the *SetFrameRate* function before saving the file. Note that in such a case the you must compute the frame rate yourself.

The frame rate information is irrelevant when the file format does not support sequences of buffers (for example BMP or TIFF formats).

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::GetBytesPerPixel

int **GetBytesPerPixel()**;

Remarks

Gets the number of bytes required to store a single buffer element of all the buffer resources.

You can only read the value of this attribute after calling the Create method.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::GetPitch

int **GetPitch()**;

Remarks

Gets the number of bytes between two consecutive lines of all the buffer resources. This is usually equal to the number of bytes per line, with possible exceptions for buffers located in video memory.

You can only read the value of this attribute after calling the Create method.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::GetBufName

const char ***GetBufName()**;

Remarks

Gets the name of a buffer object that is shared between multiple processes. If the SapBuffer object was not created using one of the constructors with shared buffers, the value of this attribute is an empty string.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::GetHandles

virtual CORHANDLE ***GetHandles**();

Remarks

Gets the low-level Sapera handles of all the buffer resources, which you may then use from low-level Sapera functionality. The handle is only valid after you call the Create method.

See the *Sapera LT Basic Modules Reference Manual* for details on low-level Sapera functionality.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::GetHandle, SapBuffer::operator[]

virtual CORHANDLE **GetHandle**();

virtual CORHANDLE **GetHandle**(int *index*);

virtual CORHANDLE **operator[]** (int *index*);

Parameters

index Index of the required buffer resource handle

Remarks

Returns the low-level Sapera handle of the specified buffer resource, which you may then use from low-level Sapera functionality. If no index is specified, the current buffer index is assumed. The handle is only valid after you call the Create method.

See the *Sapera LT Basic Modules Reference Manual* for details on low-level Sapera functionality.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::GetIndex, SapBuffer::SetIndex

virtual int **GetIndex**();

virtual BOOL **SetIndex**(int *index*);

Parameters

index Buffer resource index

Remarks

Gets/sets the index of the current buffer. The value of this attribute is set to the last buffer resource after calling the Create method. It is then automatically set by the SapTransfer class to the last acquired buffer through the Next method.

Note that all methods that access an individual buffer resource in the SapBuffer class use the current index when none is specified.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::ResetIndex

virtual void **ResetIndex**();

Parameters

index Buffer resource index

Remarks

Initializes the current buffer index to the last buffer resource, so that it will be equal to 0 after the next call to the Next method (from the SapTransfer class). This means that the first buffer resource will then be the current one.

Note that ResetIndex may be called automatically by the SapTransfer::Init method, if you set its optional argument to TRUE.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::Next

virtual void **Next**();

Remarks

Increments the current buffer index. The SapTransfer class calls Next each time an image is acquired to a buffer. The index wraps around to 0 when it reaches the end of the resource array. It always points to the last acquired buffer.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer:: Clear

virtual BOOL **Clear**();

virtual BOOL **Clear**(int *index*);

virtual BOOL **Clear**(SapData *value*);

virtual BOOL **Clear**(int *index,* SapData *value*);

Parameters

index Buffer resource index

value New value for all buffer elements. See the SapData Class and its derived classes for more details.

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Clears the content of a specified buffer resource in the array. If no value is specified, then black (usually 0) is assumed. If no index is specified, all buffers are cleared.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::Load

virtual BOOL **Load**(const char **fileName*, int *bufIndex*, int *numBuffers* = 0, int *frameIndex* = 0, const char **options* = "-format auto");

Parameters

fileName	Name of the image file to load	
bufIndex	Index of the buffer (or first buffer) in which to load, where –1 is equivalent to the current index.	
numBuffers	Maximum number of buffers to load when the file contains a sequence, where a value of 0 is equivalent to the number of buffers in the current object.	
frameIndex	Index of first image frame to load when the file contains a sequence	
options	String containing the loading options. The following are supported:	
	"-format bmp"	Window bitmap format
	"-format tiff"	TIFF format
	"-format jpeg"	JPEG format
	"-format jpeg_2000-component [value] "	JPEG 2000 format. When loading into a monochrome buffer, specify which color component to load (0 for red, 1 for green, 2 for blue); otherwise this argument is ignored.
	"-format crc"	DALSA proprietary format
	"-format raw -width [value]-height [value] -o [offset] "	Raw data format. You must specify the image width and height, as well as the offset of image data from the beginning of the file.
	"-format avi"	AVI image sequence format
	"-format auto"	Automatic format detection

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Loads an image file into the current buffer. If no *options* are specified, the format is automatically detected.

If the format is AVI, you may use *frameIndex* to specify the first frame to load from the file. If *numBuffers* is 0, the number of frames loaded will not exceed the buffer count.

If the buffer object was constructed using the same *fileName* (see the SapBuffer constructor), no data conversion will be performed since the buffer is compatible with the file.

However, if the buffer was not constructed this way, you must first use the SetParametersFromFile method to make certain that the buffer object is compatible with the file.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::Save

virtual BOOL **Save**(const char **fileName*, const char **options*, int *bufIndex* = -1, int *numBuffers* = 0);

Parameters

fileName	Name of the image file to save	
options	String containing the saving options. The following are supported:	
	"-format bmp"	Window bitmap format
	"-format tiff -compression [none/rle/lzw/jpeg] -quality [value]"	TIFF format. Compression may be set to none, run-length encoding, Lempel-Ziv-Welch, or JPEG. For the latter, you may also set a quality level.
	"-format jpeg -quality [value]"	JPEG format. The quality level may vary between 1 and 100.
	"-format jpeg_2000 -quality [value]"	JPEG 2000 format. The quality level may vary between 1 and 100, where the latter specifies lossless compression.
	"-format crc"	DALSA proprietary format
	"-format raw"	Raw data format
	"-format avi"	AVI image sequence format
bufIndex	Index of the first buffer to save when the file contains a sequence, where a value of –1 is equivalent to the first buffer. If the file contains only one image, then this is the index of the buffer resource to save and –1 is equivalent to the current index.	
numBuffers	Number of buffers to save when the file contains a sequence, where a value of 0 is equivalent to the number of buffers in the current object.	

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Saves one or more buffers to an image file.

If the format is AVI, use the *bufIndex* and *numBuffers* arguments to specify the first buffer and the number of buffers to save. When saving to a file with any other format, the *numBuffers* argument is ignored.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::Copy

virtual BOOL **Copy**(SapBuffer **pSrc*);

virtual BOOL **Copy**(SapBuffer **pSrc*, int *srcIndex*, int *dstIndex*);

Parameters

pSrc	Buffer object to copy from
srcIndex	Source buffer resource index
dstIndex	Destination buffer resource index in the current object

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Copies the contents of a single buffer resource from a source buffer object to the current object. If no source index is specified, the current source buffer index is assumed. If no destination index is specified, the current destination buffer index is assumed.

When the source buffer is larger than the destination buffer in the current object, only the section of the source that fits into the destination is copied.

If the source and destination buffer objects have different formats, automatic data conversion takes place whenever possible.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::CopyAll

virtual BOOL **CopyAll**(SapBuffer **pSrc*);

Parameters

pSrc	Buffer object to copy from

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Copies the contents of all buffer resources from a source buffer object to the current object. If the two have different buffer counts, the smaller of the two counts is used.

If the source and destination buffer objects have different formats, automatic data conversion takes place whenever possible.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::SplitComponents

virtual BOOL **SplitComponents**(SapBuffer **pSrc*,);

virtual BOOL **SplitComponents**(SapBuffer **pSrc*, int *srcIndex*);

Parameters

pSrc Source color buffer object

srcIndex Source buffer resource index

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Splits the color buffer resource at *srcIndex* into its individual monochrome components in the first three buffer resources of the current object. If no source buffer index is specified, the current value is assumed.

The destination buffer dimensions (in the current object) must be equal to or larger than the source buffer object dimensions. The input buffer format can be either RGB or YUV. See the SapBuffer constructor for a list of valid RGB and YUV formats.

If the input buffer format is RGB, then the three output buffer resources will contain the red, green, and blue components, respectively. If the input buffer format is YUV, then the three output buffer resources will contain the Y, U, and V components, respectively.

If individual color components have 8-bits or less, then the output format (in the current buffer object) must be SapFormatMono8. If color components have more than 8-bits, then the output format must be SapFormatMono16.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::MergeComponents

virtual BOOL **MergeComponents**(SapBuffer **pSrc*,);

virtual BOOL **MergeComponents**(SapBuffer **pSrc*, int *dstIndex*);

Parameters

pSrc Source monochrome buffer object

dstIndex Destination buffer resource index

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Combines the individual monochrome components from the first three buffer resources of the source object into the color buffer resource at *dstIndex* in the current object. If no destination buffer index is specified, the current

value is assumed.

The destination and source buffer dimensions must be equal. The output buffer format can be either RGB or YUV. See the SapBuffer constructor for a list of valid RGB and YUV formats.

If the output buffer format is RGB, then the three input buffer resources must contain the red, green, and blue components, respectively. If the output buffer format is YUV, then the three input buffer resources must contain the Y, U, and V components, respectively.

If individual color components have 8-bits or less, then the input format must be SapFormatMono8. If color components have more than 8-bits, then the input format must be SapFormatMono16.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::Read

virtual BOOL **Read**(int *offset*, int *numElements*, void **pData*);

virtual BOOL **Read**(int *index*, int *offset*, int *numElements*, void **pData*);

Parameters

offset	Starting position within the buffer (in pixels)
numElements	Number of pixels to read
pData	Destination memory area for pixel values
index	Buffer resource index

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Reads a consecutive series of elements (pixels) from a buffer resource, ignoring line boundaries.

For 1-bit data buffers, the offset must be a multiple of 8, and the memory area must have at least ((*numElements* + 7) >> 3) bytes.

For buffer formats other than 1-bit, the memory area must have a number of bytes larger than or equal to *numElements* times the value returned by the GetBytesPerPixel method.

If no buffer index is specified, the current index is assumed.

Reading elements from video memory buffers may be very slow.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::ReadElement

virtual BOOL **ReadElement**(int *x*, int *y*, void **pData*);

virtual BOOL **ReadElement**(int *index*, int *x*, int *y*, void **pData*);

virtual BOOL **ReadElement**(int *x*, int *y*, SapData **pData*);

virtual BOOL **ReadElement**(int *index*, int *x*, int *y*, SapData **pData*);

Parameters

x	Horizontal position
y	Vertical position
pData	Pointer to a destination memory area for the pixel value, or to one of the following wrapper classes for Sapera data elements: SapDataMono Class, SapDataRGB Class, SapDataRGBA Class, SapDataYUV Class, SapDataFloat Class, SapDataPoint Class, and SapDataFPoint Class
index	Buffer resource index

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Reads a single element (pixel) from a buffer resource.

If using one of the first two forms of ReadElement, the memory area must have a number of bytes larger than or equal to the value returned by the GetBytesPerPixel method.

If no buffer index is specified, the current index is assumed.

Reading elements from video memory buffers may be very slow.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::ReadLine

virtual BOOL **ReadLine**(int *x1*, int *y1*, int *x2*, int *y2*, void **pData,* int **numRead*);

virtual BOOL **ReadLine**(int *index*, int *x1*, int *y1*, int *x2*, int *y2*, void **pData,* int **numRead*);

Parameters

x1	Starting horizontal position
y1	Starting vertical position
x2	Ending horizontal position
y2	Ending vertical position
pData	Destination memory area for pixel values
numRead	Returns the number of pixels read along the line
index	Buffer resource index

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Reads one line of buffer elements, from position (*x1*,*y1*) to position (*x2*,*y2*). Diagonal lines are supported.

The memory area must have a number of bytes larger than or equal to the line length times the value returned by the GetBytesPerPixel method.

If no buffer index is specified, the current index is assumed.

ReadLine does not support 1-bit buffers.

Reading elements from video memory buffers may be very slow.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::ReadRect

virtual BOOL **ReadRect**(int *x*, int *y*, int *width*, int *height*, void **pData*);

virtual BOOL **ReadRect**(int *index*, int *x*, int *y*, int *width*, int *height*, void **pData*);

Parameters

x	Left coordinate of rectangle origin
y	Top coordinate of rectangle origin
width	Rectangle width
height	Rectangle height
pData	Destination memory area for pixel values
index	Buffer resource index

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Reads a rectangular region of elements (pixels) from a buffer resource.

For 1-bit data buffers, *x* and *width* must be multiples of 8, and the memory area must have at least ((*numElements* + 7) >> 3) bytes.

For buffer formats other than 1-bit, the memory area must have a number of bytes larger than or equal to *numElements* times the value returned by the GetBytesPerPixel method.

If no buffer index is specified, the current index is assumed.

Reading elements from video memory buffers may be very slow.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::Write

virtual BOOL **Write**(int *offset*, int *numElements*, void **pData*);

virtual BOOL **Write**(int *index*, int *offset*, int *numElements*, void **pData*);

Parameters

offset	Starting position within the buffer (in pixels)
numElements	Number of pixels to write
pData	Source memory area for pixel values
index	Buffer resource index

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Writes a consecutive series of elements (pixels) to a buffer resource, ignoring line boundaries.

For 1-bit data buffers, the offset must be a multiple of 8, and the memory area must have at least ((*numElements* + 7) >> 3) bytes.

For buffer formats other than 1-bit, the memory area must have a number of bytes of at least *numElements* times the value returned by the GetBytesPerPixel method.

If no buffer index is specified, the current index is assumed.

Writing elements to video memory buffers may be very slow.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::WriteElement

virtual BOOL **WriteElement**(int *x*, int *y*, const void **pData*);

virtual BOOL **WriteElement**(int *index*, int *x*, int *y*, const void **pData*);

virtual BOOL **WriteElement**(int *x*, int *y*, SapData *data*);

virtual BOOL **WriteElement**(int *index*, int *x*, int *y*, SapData *data*);

Parameters

x	Horizontal position
y	Vertical position
pData	Pointer to a source memory area for the pixel value
data	Pixel value represented by one of the following wrapper classes for Sapera data elements: SapDataMono, SapDataRGB, SapDataYUV, SapDataFloat, SapDataPoint, and SapDataFPoint
index	Buffer resource index

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Writes a single element (pixel) to a buffer resource.

If using one of the first two forms of WriteElement, the memory area must have a number of bytes equal or larger than the value returned by the GetBytesPerPixel method.

If no buffer index is specified, the current index is assumed.

Writing elements to video memory buffers may be very slow.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::WriteLine

virtual BOOL **WriteLine**(int *x1*, int *y1*, int *x2*, int *y2*, const void **pData,* int **numWritten*);

virtual BOOL **WriteLine**(int *index*, int *x1*, int *y1*, int *x2*, int *y2*, const void **pData,* int **numWritten*);

Parameters

x1	Starting horizontal position
y1	Starting vertical position
x2	Ending horizontal position
y2	Ending vertical position
pData	Source memory area for pixel values
numWritten	Returns the number of pixels written along the line
index	Buffer resource index

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Writes one line of buffer elements, from position (*x1*,*y1*) to position (*x2*,*y2*). Diagonal lines are supported.

The memory area must have a number of bytes larger than or equal to the line length times the value returned by the GetBytesPerPixel method.

If no buffer index is specified, the current index is assumed.

WriteLine does not support 1-bit buffers.

Writing elements to video memory buffers may be very slow.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::WriteRect

virtual BOOL **WriteRect**(int *x*, int *y*, int *width*, int *height*, const void **pData*);

virtual BOOL **WriteRect**(int *index*, int *x*, int *y*, int *width*, int *height*, const void **pData*);

Parameters

x	Left coordinate of rectangle origin
y	Top coordinate of rectangle origin
width	Rectangle width
height	Rectangle height
pData	Source memory area for pixel values
index	Buffer resource index

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Writes a rectangular region of elements (pixels) to a buffer resource.

For 1-bit data buffers, *x* and *width* must be multiples of 8, and the memory area must have at least ((*numElements* + 7) >> 3) bytes.

For buffer formats other than 1-bit, the memory area must have a number of bytes larger than or equal to *numElements* times the value returned by the GetBytesPerPixel method.

If no buffer index is specified, the current index is assumed.

Writing elements to video memory buffers may be very slow.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::GetState

virtual BOOL **GetState**(SapBuffer::State **pState*);

virtual BOOL **GetState**(int *index*, SapBuffer::State **pState*);

Parameters

pState Pointer to the returned buffer state, which may be one of the following:

SapBuffer::StateEmpty	The buffer is ready to receive new data
SapBuffer::StateFull	The buffer contains unprocessed data
SapBuffer::StateOverflow	The buffer contains incorrect data due to insufficient hardware bandwidth. This state may only occurs together when StateEmpty

is active (the two values are combined using bitwise OR).

index Buffer resource index

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the current buffer state that indicates whether the specified buffer is ready to accept a new image, or currently contains unprocessed data.

If no buffer index is specified, the current index is assumed.

Note that Sapera automatically manages the buffer state by default, so that you rarely have to call GetState directly. If you wish to perform this management yourself, you must first call SapTransfer::SetAutoEmpty.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::SetState

virtual BOOL **SetState**(SapBuffer::State *state*);

virtual BOOL **SetState**(int *index*, SapBuffer::State *state*);

Parameters

state New state for the buffer resource. See the SapBuffer::GetState method for possible values.

index Buffer resource index

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Sets the current buffer state that indicates whether the specified buffer is ready to accept a new image, or currently contains unprocessed data.

If no buffer index is specified, the current index is assumed.

Note that Sapera automatically manages the buffer state by default, so that you rarely have to call SetState directly. If you wish to perform this management yourself, you must first call SapTransfer::SetAutoEmpty.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::SetAllState

virtual BOOL **SetAllState**(SapBuffer::State *state*);

Parameters

state New state for the buffer resources. See the SapBuffer::GetState method for possible values.

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Sets the current state for all buffer resources in the current object. See also the SapBuffer::SetState method.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::GetPage

virtual BOOL **GetPage**(int **pPage*);

virtual BOOL **GetPage**(int *index*, int **pPage*);

Parameters

pPage Pointer to the returned page number

index Buffer resource index

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the active page (or plane) of the current buffer resource.

This applies only to buffer types for which pixel data is stored in separate planes, instead of being packed together. For example, 8-bit RGB planar (SapFormatRGBP8) or 8-bit HSI planar (SapFormatHSIP8).

The active page only affects image display. For example, if the image format is 8-bit RGB planar and the page index is 0, then the red component will be displayed. If the index is 1 or 2, then the green and blue components will be displayed, respectively.

If no buffer index is specified, the current index is assumed.

SapBuffer | SapBuffer Class Members | Hierarchy Charts

SapBuffer::SetPage

virtual BOOL **SetPage**(int *page*);

virtual BOOL **SetPage**(int *index*, int *page*);

Parameters

page New page number for the buffer resource

index Buffer resource index

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Sets the active page (or plane) of the current buffer resource.

This applies only to buffer types for which pixel data is stored in separate planes, instead of being packed together. For example, 8-bit RGB planar (SapFormatRGBP8) or 8-bit HSI planar (SapFormatHSIP8).

The active page only affects image display. For example, if the image format is 8-bit RGB planar and the page index is 0, then the red component will be displayed. If the index is 1 or 2, then the green and blue components will be displayed, respectively.

If no buffer index is specified, the current index is assumed.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::SetAllPage

virtual BOOL **SetAllPage**(int *page*);

Parameters

page New page number for the buffer resources

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Sets the active page (or plane) for all buffer resources in the current object. See also the SapBuffer::SetPage method.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::GetAddress

virtual BOOL **GetAddress**(void ***pData*);

virtual BOOL **GetAddress**(void **virtualBaseAddress*, void ***pData*);

virtual BOOL **GetAddress**(int *index*, void ***pData*);

virtual BOOL **GetAddress**(int *index*, void **virtualBaseAddress*, void ***pData*);

virtual BOOL **GetAddress**(int *offset*, int *size*, void ***pData*);

virtual BOOL **GetAddress**(int *offset*, int *size*, void **virtualBaseAddress*, void ***pData*);

virtual BOOL **GetAddress**(int *index*, int *offset*, int *size*, void ***pData*);

virtual BOOL **GetAddress**(int *index*, int *offset*, int *size*, void **virtualBaseAddress*, void ***pData*);

Parameters

virtualBaseAddress	Starting address of a memory area already reserved by the application
pData	Pointer to returned buffer data address
index	Buffer resource index
offset	Byte offset from beginning of buffer data for partial mapping
size	Number of bytes of buffer data to access for partial mapping

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the virtual address where buffer data is stored. Call GetAddress when you need to process buffers in the application itself. Since the SapBuffer Read and Write methods are too slow for this purpose, you need direct access through a pointer. In order to correctly interpret the raw data, you also need to use some or all of the following methods: GetWidth, GetHeight, GetFormat, GetPixelDepth, GetBytesPerPixel, and GetPitch.

Accessing buffer data in video memory may be very slow. In this case, you must call the ReleaseAddress method as soon as possible when you are finished, since getting the address prevents the display hardware from accessing buffer data. This may result in image display problems.

When dealing with buffers that are TypeUnmapped or TypeScatterGatherUnmapped, you should call the ReleaseAddress method as soon as possible when you are done. Getting the data address causes the actual physical to virtual memory mapping to occur. Releasing the address ends the memory mapping and may prevent exhaustion of virtual memory resources in the operating system.

When dealing with very large buffers, you may want to map the buffer data area one section at a time, since fully mapping a very large amount of memory can consume a large amount of system resources. In this case, use the offset and size arguments to specific the partial area to map, and call the ReleaseAddress method before mapping another section.

If you need control over the addresses where the buffer mapping occurs, then use the *virtualBaseAddress* argument. It allows you to specify an address of memory that has already been reserved by the application as the base address for memory mapping.

If no buffer index is specified, the current index is assumed.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::ReleaseAddress

virtual BOOL **ReleaseAddress**(void **pData*);

virtual BOOL **ReleaseAddress**(int *index*, void **pData* = NULL);

Parameters

pData Buffer data address to release

index Buffer resource index

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Ends direct buffer data access through a pointer.

When dealing with buffers located in video memory, you must call ReleaseAddress as soon as possible after GetAddress; otherwise, you may encounter image display problems, since getting the address prevents the display hardware from accessing buffer data.

When dealing with buffers that are TypeUnmapped or TypeScatterGatherUnmapped, you should call ReleaseAddress as soon as possible when you are finished with direct data access. Calling the GetAddress method causes the actual physical to virtual memory mapping to occur. Releasing the address ends the memory mapping and may prevent exhaustion of virtual memory resources in the operating system.

If no buffer index is specified, the current index is assumed.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::IsMapped

virtual BOOL **IsMapped**();

virtual BOOL **IsMapped**(int *index*);

Parameters

index Buffer resource index

Return Value

Returns TRUE if there currently exists a valid virtual memory address for the specified buffer resource, FALSE otherwise

Remarks

This method is only relevant for buffers that are TypeUnmapped or TypeScatterGatherUnmapped. In this case, the GetAddress method sets up a valid virtual address mapping, and ReleaseAddress ends it. For all other buffer types, it always returns TRUE.

If no buffer index is specified, the current index is assumed.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::GetCounterStamp

virtual BOOL **GetCounterStamp**(int **pCounterStamp*);

virtual BOOL **GetCounterStamp** (int *index*, int **pCounterStamp*);

Parameters

pCounterStamp Pointer to the returned counter value for the specified buffer resource

index Buffer resource index

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets a unique value associated with a buffer resource. This value is normally expressed in microseconds. It has no meaning by itself; however, subtracting counter stamp values for two buffer resources gives the amount of time elapsed between a common reference point for their respective data transfers.

The counter stamp value may also be expressed in other units. See the SapXferPair::GetCounterStampTimeBase, SapXferPair::SetCounterStampTimeBase method for details.

Note that some transfer devices do not support this feature.

If no buffer index is specified, the current index is assumed.

SapBuffer::GetSpaceUsed

virtual BOOL **GetSpaceUsed**(int **pSpaceUsed*);

virtual BOOL **GetSpaceUsed** (int *index*, int **pSpaceUsed*);

Parameters

pSpaceUsed	Pointer to the returned space used value for the specified buffer resource
index	Buffer resource index

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the actual number of data bytes stored in a buffer resource after acquiring an image. This is normally the same value that you would get by multiplying the values returned by the GetWidth, GetHeight, and GetBytesPerPixel methods. However, when acquiring variable-length images, this value may be smaller.

If no buffer index is specified, the current index is assumed.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::Register

virtual BOOL **Register**(const char **name*);

Parameters

name	Name to assign to the current SapBuffer object

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Registers the current SapBuffer object to enable remote access. Using the supplied *name* argument, Register stores unique names for all buffer resources into an internal database. A Sapera application running on a remote server may then access these resources by creating a SapBufferRemote object with the *name* used in this method.

You may call the Unregister method after the remote server has finished accessing the buffer resources.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::Unregister

virtual BOOL Unregister();

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Disable remote access for all buffer resources in the current SapBuffer object. Unregister must be called after a Sapera application running on a remote server has finished accessing these resources. See the SapBuffer::Register method for more information.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::GetCapability

virtual BOOL **GetCapability**(int *cap*, void **pValue*);

virtual BOOL **GetCapability**(int *index*, int *cap*, void **pValue*);

Parameters

cap	Low-level Sapera C library capability to read
pValue	Pointer to capability value to read back
index	Buffer resource index

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

This method allows direct read access to low-level Sapera C library capabilities for the buffer module. It needs a pointer to a memory area large enough to receive the capability value, which is usually a 32-bit integer. If no index is specified, the current buffer index is assumed.

Note that you will rarely need to use GetCapability. The Class already uses important capabilities internally for self-configuration and validation.

See the *Sapera LT Basic Modules Reference Manual* for a description of all capabilities and their possible values.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBuffer::GetParameter, SapBuffer::SetParameter

virtual BOOL **GetParameter**(int *param*, void **pValue*);

virtual BOOL **GetParameter**(int *index*, int *param*, void **pValue*);

virtual BOOL **SetParameter**(int *param*, int *value*);

virtual BOOL **SetParameter**(int *index*, int *param*, int *value*);

virtual BOOL **SetParameter**(int *param*, void **pValue*);

virtual BOOL **SetParameter**(int *index*, int *param*, void **pValue*);

Parameters

param	Low-level Sapera C library parameter to read or write
pValue	Pointer to parameter value to read back or to write
index	Buffer resource index
value	New parameter value to write

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

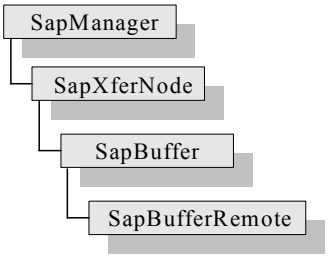
These methods allow direct read/write access to low-level Sapera C library parameters for the buffer module. The GetParameter method needs a pointer to a memory area large enough to receive the parameter value, which is usually a 32-bit integer. The first form of SetParameter accepts a 32-bit value for the new value. The second form takes a pointer to the new value, and is required when the parameter uses more than 32 bits of storage. If no index is specified, the current buffer index is assumed.

Note that you will rarely need to use these methods. You should first make certain that what you need is not already supported through the Class. Also, directly setting parameter values may interfere with the correct operation of the class.

See the *Sapera LT Basic Modules Reference Manual* for a description of all parameters and their possible values.

SapBuffer | SapBuffer Class Members | Basic Class Hierarchy Chart

SapBufferRemote



The SapBufferRemote Class is used to create a “wrapper” over an existing SapBuffer object on a remote server.

If a SapBuffer object exists on a remote server, and has been registered using the SapBuffer::Register method, then a SapBufferRemote object may be created locally to access the functionality of the remote object.

One typical usage of this class is data exchange between two different servers.

#include <SapClassBasic.h>

SapBufferRemote Class Members | Base Class: SapManager | Basic Class Hierarchy Chart

SapBufferRemote Class Members

Construction	
SapBufferRemote	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources

SapBufferRemote | Base Class: SapManager | Basic Class Hierarchy Chart

Member Functions

The following functions are members of the SapBufferRemote Class.

SapBufferRemote::SapBufferRemote

```
SapBufferRemote(
  SapLocation loc,
  const char *name,
  int startIndex = 0,
  int count = 1
);
```

Parameters

loc	SapLocation object specifying the server where the remote buffer resources are located. The resource index of the location object is ignored.
name	The registered name of the remote object. This name must match the one used when registering the remote buffer with the SapBuffer::Register method.
startIndex	Index of the first buffer resource in the remote buffer object
count	Number of buffer resources in the remote buffer object

Remarks

The SapBufferRemote object allows creation of a "wrapper" over an existing remote buffer object. This means that the actual low-level resources are created remotely and then made available through the SapBuffer::Register method. Their values are then encapsulated in the local object. From then on, you may use the buffer object as if it had been created locally.

You may specify a subset of the remote buffer resources using the *startIndex* and *count* arguments.

The constructor does not actually access the low-level Sapera resources. To do this, you must call the Create method.

See to the SapBuffer::Register method for more details on how to register the remote buffer object.

SapBufferRemote | SapBufferRemote Class Members | Basic Class Hierarchy Chart

SapBufferRemote::Create

virtual BOOL **Create**();

Return Value

Returns TRUE if the object was successfully created.

Remarks

Gets access to all buffer resources of the remote buffer object.

SapBufferRemote | SapBufferRemote Class Members | Basic Class Hierarchy Chart

SapBufferRemote::Destroy

virtual BOOL **Destroy**();

Return Value

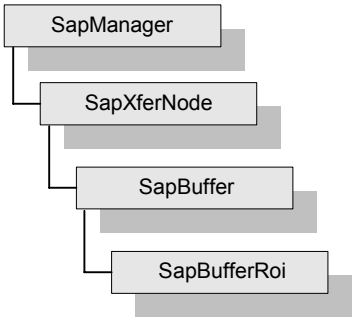
Returns TRUE if the object was successfully destroyed.

Remarks

Release access to all the buffer resources of the remote buffer object.

SapBufferRemote | SapBufferRemote Class Members | Basic Class Hierarchy Chart

SapBufferRoi



The purpose of the SapBufferRoi Class is to create a rectangular Region of Interest (ROI) inside an existing SapBuffer object. The ROI has the same origin and dimensions for all buffer resources in the object.

You may create multiple instances of this class using the same SapBuffer as a parent; however, the acquisition hardware dictates the number of maximum simultaneous ROIs when acquiring images.

One typical usage of this class is reducing acquisition bandwidth requirements when only a subset of an image is needed.

#include <SapClassBasic.h>

SapBufferRoi Class Members | Base Class: SapManager | Basic Class Hierarchy Chart

SapBufferRoi Class Members

Construction	
SapBufferRoi	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Attributes	
GetParent	Gets/sets the parent SapBuffer object for the ROI

SetParent	
GetRoot	Gets the topmost SapBuffer object for the ROI
GetXMin SetXMin	Gets/sets the left origin for the ROI, relative to the parent object
GetYMin SetYMin	Gets/sets the top origin for the ROI, relative to the parent object
GetWidth SetWidth	Gets/sets the width (in pixels) for the ROI
GetHeight SetHeight	Gets/sets the height (in pixels) for the ROI
SetRoi	Sets the ROI origin and dimensions in one step
ResetRoi	Sets the ROI origin and dimensions to default values
GetTrash	Returns the low-level Sapera handle of the trash buffer resource, if any

SapBufferRoi | Base Class: SapManager | Basic Class Hierarchy Chart

Member Functions

The following functions are members of the SapBufferRoi Class.

SapBufferRoi::SapBufferRoi

```
SapBufferRoi(
   SapBuffer *pParent,
   int xmin = 0,
   int ymin = 0,
   int width = -1,
   int height = -1
);
```

Parameters

pParent	SapBuffer object that represents the parent for the current SapBufferRoi object
xmin	Left origin for the ROI, relative to the parent object
ymin	Top origin for the ROI, relative to the parent object
width	Width (in pixels) of the ROI
height	Height (in lines) of the ROI

Remarks

The SapBufferRoi constructor sets up a rectangular Region of Interest (ROI) inside the SapBuffer object identified by *pParent*. This ROI has the specified origin and dimensions, up to the whole area of the parent object.

A value of –1 for the width/height means that the ROI should have the same width/height as the parent buffer.

The constructor does not actually create the low-level Sapera resources. To do this, you must call the Create method.

SapBufferRoi | SapBufferRoi Class Members | Basic Class Hierarchy Chart

SapBufferRoi::Create

virtual BOOL **Create**();

Return Value

Returns TRUE if the object was successfully created, FALSE otherwise

Remarks

Creates all the low-level Sapera resources needed by the current object. Always call this method before SapTransfer::Create.

SapBufferRoi | SapBufferRoi Class Members | Basic Class Hierarchy Chart

SapBufferRoi::Destroy

virtual BOOL **Destroy**();

Return Value

Returns TRUE if the object was successfully destroyed, FALSE otherwise

Remarks

Destroys all low-level Sapera resources used by the current object. Always call this method after SapTransfer::Destroy.

SapBufferRoi | SapBufferRoi Class Members | Basic Class Hierarchy Chart

SapBufferRoi::GetParent, SapBuffer::SetParent

SapBuffer ***GetParent**();

virtual void **SetParent**(SapBuffer **pParent*);

Remarks

Gets/sets the parent buffer object for the ROI. Note that you can only call SetParent before the Create method.

SapBufferRoi | SapBufferRoi Class Members | Basic Class Hierarchy Chart

SapBufferRoi::GetRoot

SapBuffer ***GetRoot**();

Remarks

Gets the topmost SapBuffer object for the ROI.

When there is a one-level ROI hierarchy below the topmost object, then the returned value is the same as for the GetParent method.

When there is a multi-level ROI hierarchy below the topmost object, then the returned value is the equivalent of going up the ROI tree by calling GetParent repeatedly until we reach the top.

SapBufferRoi | SapBufferRoi Class Members | Basic Class Hierarchy Chart

SapBufferRoi::GetXMin, SapBufferRoi::SetXMin

int **GetXMin**();

virtual void **SetXMin**(int *xmin*);

Remarks

Gets/sets the left origin for the ROI, relative to the parent object. The initial value of this attribute is 0 if it was not specified in the constructor.

You can only call SetXMin before the Create method.

SapBufferRoi | SapBufferRoi Class Members | Basic Class Hierarchy Chart

SapBufferRoi::GetYMin, SapBufferRoi::SetYMin

int **GetYMin**();

virtual void **SetYMin**(int *ymin*);

Remarks

Gets/sets the top origin for the ROI, relative to the parent object. The initial value of this attribute is 0 if it was not specified in the constructor.

You can only call SetYMin before the Create method.

SapBufferRoi | SapBufferRoi Class Members | Basic Class Hierarchy Chart

SapBufferRoi::GetWidth, SapBufferRoi::SetWidth

int **GetWidth**();

virtual void **SetWidth**(int *width*);

Remarks

Gets/sets the width (in pixels) for the ROI. If it has not been specified in the constructor, the value of this attribute is set to the parent buffer width when calling the Create method.

You can only call SetWidth before the Create method.

SapBufferRoi | SapBufferRoi Class Members | Basic Class Hierarchy Chart

SapBufferRoi::GetHeight, SapBufferRoi::SetHeight

int **GetHeight**();

virtual void **SetHeight**(int *height*);

Remarks

Gets/sets the height (in lines) for the ROI. If it has not been specified in the constructor, the value of this attribute is set to the parent buffer height when calling the Create method.

You can only call SetHeight before the Create method.

SapBufferRoi | SapBufferRoi Class Members | Basic Class Hierarchy Chart

SapBufferRoi::SetRoi

virtual BOOL **SetRoi**(int *xmin*, int *ymin,* int *width*, int *height*);

virtual BOOL **SetRoi**(RECT **pRect*);

Parameters

xmin	Left origin for the ROI, relative to the parent object
ymin	Top origin for the ROI, relative to the parent object
width	Width (in pixels) of the ROI
height	Height (in lines) of the ROI
pRect	Pointer to a Windows RECT structure that specifies the four corners of the ROI

Remarks

Sets the ROI origin and dimensions in one step. You can only call SetRoi before the Create method.

SapBufferRoi | SapBufferRoi Class Members | Basic Class Hierarchy Chart

SapBufferRoi::ResetRoi

virtual BOOL **ResetRoi**();

Remarks

Sets the ROI origin and dimensions to default values corresponding to the whole buffer area in the parent object. You can only call ResetRoi before the Create method.

SapBufferRoi | SapBufferRoi Class Members | Basic Class Hierarchy Chart

SapBufferRoi::GetTrash

virtual CORBUFFER **GetTrash**();

Remarks

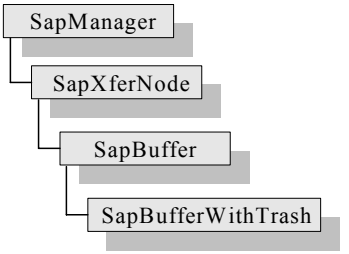
If the current object has a SapBufferWithTrash parent object, then this method returns the low-level Sapera handle of the ROI trash buffer resource, which you may then use from the low-level Sapera functionality.

Note that the handle is only valid after you call the Create method.

See the *Sapera LT Basic Modules Reference Manual* for details on low-level Sapera functionality.

SapBufferRoi | SapBufferRoi Class Members | Basic Class Hierarchy Chart

SapBufferWithTrash



The SapBufferWithTrash Class is derived from SapBuffer. It creates an additional resource called the trash buffer used when transferring data in real-time applications.

Trash buffer is an emergency buffer used by the SapTransfer Class when the data transfer is faster than a processing task performed on the buffers. When processing is not fast enough to keep up with the incoming data, images are transferred temporarily into the trash buffer until stability is reestablished.

#include <SapClassBasic.h>

SapBufferWithTrash Class Members | Base Class: SapManager | Basic Class Hierarchy Chart

SapBufferWithTrash Class Members

Construction	
SapBufferWithTrash	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Attributes	
GetTrashType, SetTrashType	Gets/sets the buffer type for the trash buffer resource only
GetTrash	Returns the low-level Sapera handle of the trash buffer resource

SapBufferWithTrash | Base Class: SapManager | Basic Class Hierarchy Chart

Member Functions

The following functions are members of the SapBufferWithTrash Class.

SapBufferWithTrash::SapBufferWithTrash

```
SapBufferWithTrash(
  int count = 2,
  int width = SapDefWidth,
  int height = SapDefHeight,
  SapFormat format = SapDefFormat,
  SapBuffer::Type type = SapDefBufferType,
  SapLocation loc = SapLocation::ServerSystem
);
SapBufferWithTrash(
  int count,
  int *physAddress
  int width = SapDefWidth,
  int height = SapDefHeight,
  SapFormat format = SapDefFormat,
  SapBuffer::Type type = SapBuffer::TypeContiguous,
);
SapBufferWithTrash (
  int count,
  void **virtAddress
  int width = SapDefWidth,
  int height = SapDefHeight,
  SapFormat format = SapDefFormat,
  SapBuffer::Type type = SapBuffer::TypeScatterGather,
);
SapBufferWithTrash (
  int count,
  SapXferNode *pSrcNode,
  SapBuffer::Type type = SapDefBufferType,
  SapLocation loc =  SapLocation::ServerSystem
);
SapBufferWithTrash (
  int count,
  const char *bufName
  int width = SapDefWidth,
  int height = SapDefHeight,
  SapFormat format = SapDefFormat,
```

```
  SapBuffer::Type type = SapBuffer::TypeScatterGather,
  SapLocation loc =  SapLocation::ServerSystem
);
SapBufferWithTrash (
  int count,
  const char *bufName
  SapXferNode *pSrcNode,
  SapBuffer::Type type = SapDefBufferType,
  SapLocation loc =  SapLocation::ServerSystem
);
SapBufferWithTrash (
  const char *bufName
  int startIndex
  int count,
  SapBuffer::Type type = SapBuffer::TypeVirtual,
  SapLocation loc =  SapLocation::ServerSystem
);
```

Parameters

See the SapBuffer Class constructor for a description of the parameters

Remarks

Derived from SapBuffer, the SapBufferWithTrash object contains an additional resource called the trash buffer that has the same parameters (width, height, format, and type) as the other buffer resources.

The *count* argument does not include the trash buffer. Its value cannot be smaller than 2.

The constructor does not actually create the low-level Sapera resources. To do this, you must call the Create method.

SapBufferWithTrash | SapBufferWithTrash Class Members | Basic Class Hierarchy Chart

SapBufferWithTrash::Create

```
virtual BOOL Create();
```

Return Value

Returns TRUE if the object was successfully created, FALSE otherwise

Remarks

Creates all the low-level Sapera resources needed by the SapBufferWithTrash object. Always call this method before SapTransfer::Create.

SapBufferWithTrash | SapBufferWithTrash Class Members | Basic Class Hierarchy Chart

SapBufferWithTrash::Destroy

virtual BOOL **Destroy**();

Return Value

Returns TRUE if the object was successfully destroyed, FALSE otherwise

Remarks

Destroys all low-level Sapera resources needed by the SapBufferWithTrash object. Always call this method after SapTransfer::Destroy.

SapBufferWithTrash | SapBufferWithTrash Class Members | Basic Class Hierarchy Chart

SapBufferWithTrash::GetTrashType, SapBufferWithTrash::SetTrashType

SapBuffer::Type **GetTrashType**();

virtual void **SetTrashType**(SapBuffer::Type *type*);

Remarks

Gets/sets the buffer type for the trash buffer resource only. This may be useful, for example, if the current transfer device allows the usage of dummy buffers (SapBuffer::TypeDummy) to reduce bandwidth requirements associated with trash buffer transfers.

If you do not set a value for this property, then it is set to the same value as the other buffer resources when calling the Create method

You can only call SetTrashType before the Create method. See the SapBuffer constructor for possible values for *type*.

SapBufferWithTrash | SapBufferWithTrash Class Members | Basic Class Hierarchy Chart

SapBufferWithTrash::GetTrash

virtual CORBUFFER **GetTrash**();

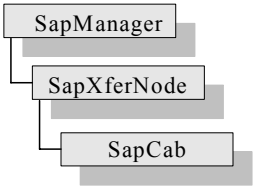
Remarks

Returns the low-level Sapera handle of the trash buffer resource, which you may then use from the low-level Sapera functionality. Note that the handle is only valid after you call the Create method.

See the *Sapera LT Basic Modules Reference Manual* for details on low-level Sapera functionality.

SapBufferWithTrash | SapBufferWithTrash Class Members | Basic Class Hierarchy Chart

SapCab



The SapCab Class includes the functionality to manipulate a CAB resource. It may be used as a destination transfer node to allow data transfer from a source node (such as acquisition buffer, DSP, Pixel Processor, or another CAB) to a CAB resource. It may also be used as a source transfer node to allow transferring data from a CAB resource to a buffer or another CAB resource.

#include <SapClassBasic.h>

SapCab Class Members | Base Class: SapManager | Basic Class Hierarchy Chart

SapCab Class Members

Construction	
SapCab	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Operations	
GetCapability	Gets the value of a low-level Sapera C library capability
GetParameter SetParameter	Gets/sets the value of a low-level Sapera C library parameter

SapCab | Base Class: SapManager | Basic Class Hierarchy Chart

Member Functions

The following functions are members of the SapCab Class.

SapCab::SapCab

```
SapCab(
  SapLocation loc = SapLocation::ServerSystem,
  SapXferNode *pSrcNode = NULL
);
SapCab(
  SapLocation loc,
  SapXferParams xferParams
);
```

Parameters

loc	SapLocation object specifying the server where the CAB resource is located and the index of the resource on this server
pSrcNode	Source node object. To ensure transfer compatibility, this object must match the source node specified when adding a transfer pair (SapXferPair) to the SapTransfer object.
xferParams	Transfer parameters of the source node. Its attributes, to ensure transfer compatibility, must match the equivalent attributes of the source node specified when adding a transfer pair (SapXferPair) to the SapTransfer object.

Remarks

The SapCab constructor does not actually create the low-level Sapera resources. To do this, you must call the Create method.

The SapCab object is used only for storing the CAB resource parameters. To transfer data to/from the CAB device, you must use the SapTransfer Class (or one of its derived classes) and specify the SapCab object as a parameter. The data transfer is then controlled by the SapTransfer Class.

SapCab | SapCab Class Members | Basic Class Hierarchy Chart

SapCab::Create

virtual BOOL **Create**();

Return Value

Returns TRUE if the object was successfully created, FALSE otherwise

Remarks

Creates all low-level Sapera resources needed by the CAB object. Always call this method before SapTransfer::Create.

SapCab | SapCab Class Members | Basic Class Hierarchy Chart

SapCab::Destroy

virtual BOOL **Destroy**();

Return Value

Returns TRUE if the object was successfully destroyed, FALSE otherwise

Remarks

Destroys all low-level Sapera resources needed by the CAB object. Always call this method after SapTransfer::Destroy.

SapCab | SapCab Class Members | Basic Class Hierarchy Chart

SapCab::GetCapability

virtual BOOL **GetCapability**(int *cap*, void **pValue*);

Parameters

param	Low-level Sapera C library capability to read
pValue	Pointer to capability value to read back

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

This method allows direct read access to low-level Sapera C library capabilities for the CAB module. It needs a pointer to a memory area large enough to receive the capability value, which is usually a 32-bit integer.

GetCapability is rarely needed. The SapCab Class already uses important capabilities internally for self-configuration and validation.

See the *CAB Programmer's Manual* for a description of all capabilities and their possible values.

SapCab | SapCab Class Members | Basic Class Hierarchy Chart

SapCab::GetParameter, SapCab::SetParameter

virtual BOOL **GetParameter**(int *param*, void **pValue*);

virtual BOOL **SetParameter**(int *param*, int *value*);

virtual BOOL **SetParameter**(int *param*, void **pValue*);

Parameters

param	Low-level Sapera C library parameter to read or write
pValue	Pointer to parameter value to read back or to write
value	New parameter value to write

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

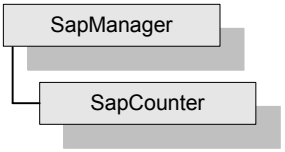
These methods allow direct read/write access to low-level Sapera C library parameters for the CAB module. The GetParameter method needs a pointer to a memory area large enough to receive the parameter value, which is usually a 32-bit integer. The first form of SetParameter accepts a 32-bit value for the new value. The second form takes a pointer to the new value, and is required when the parameter uses more than 32 bits of storage.

Note that these methods are rarely needed. First make certain that the parameter needed is not already supported through the SapCab Class. Also, directly setting parameter values may interfere with the correct operation of the class.

See the *CAB Programmer's Manual* for a description of all parameters and their possible values.

SapCab | SapCab Class Members | Basic Class Hierarchy Chart

SapCounter



The purpose of the SapCounter Class is to count events. These events can be external, such as a user supplied signal, or internal, such as a hardware clock The counter may then be used as a reference to control events, for example, change the state of a general I/O at a specific time (SapGio Class). It may also be used to timestamp acquired images (SapBuffer Class), or to monitor the progression of an application (by simply reading the counter value).

Note that acquisition devices do not all support event counting.

#include <SapClassBasic.h>

SapCounter Class Members | Base Class: SapManager | Basic Class Hierarchy Chart

SapCounter Class Members

Construction	
SapCounter	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Attributes	
GetLocation	Gets the location where the counter resource is located
SetCallbackInfo	Sets the application callback method for counter events and the associated context
GetCallback	Gets the current application callback method for counter events
GetContext	Gets the application context associated with counter events
GetBaseUnits SetBaseUnits	Gets/sets the basic counter units
GetDetectType SetDetectType	Gets/sets the detection method of events where the counter value changes

GetDirection SetDirection	Gets/sets the counting direction (increment vs decrement)
GetCount	Gets the current counter value
GetHandle	Gets the low-level Sapera handle of the counter resource
Operations	
Start	Starts the counter device
Stop	Stops the counter device
EnableCallback	Allows an application callback function to be called at specific counter events
DisableCallback	Disables calls to the application callback function
GetCapability	Gets the value of a low-level Sapera C library capability
GetParameter SetParameter	Gets/sets the value of a low-level Sapera C library parameter

SapCounter | Base Class: SapManager | Basic Class Hierarchy Chart

Member Functions

The following functions are members of the SapCounter Class.

SapCounter::SapCounter

SapCounter(
  SapLocation *loc* = SapLocation::ServerSystem,
  SapCounterCallback *pCallback* = NULL,
  void **pContext* = NULL
);

Parameters

loc	SapLocation object specifying the server where the counter resource is located and the index of the resource on this server
pCallback	Application callback function to be called each time a counter event happens. The callback function must be declared as: void MyCallback(SapCounterCallbackInfo **pInfo*);
pContext	Optional pointer to an application context to be passed to the callback function. If *pCallback* is NULL, this parameter is ignored.

Remarks

The SapCounter constructor does not actually create the low-level Sapera resources. To do this, you must call

the Create method.

Specifying a callback function in the constructor does not automatically activate it after the call to the Create method. You must subsequently call the EnableCallback method in order to be notified of counter events.

SapCounter | SapCounter Class Members | Basic Class Hierarchy Chart

SapCounter::Create

virtual BOOL **Create**();

Return Value

Returns TRUE if the object was successfully created, FALSE otherwise

Remarks

Creates all the low-level Sapera resources needed by the counter object

SapCounter | SapCounter Class Members | Basic Class Hierarchy Chart

SapCounter::Destroy

virtual BOOL **Destroy**();

Return Value

Returns TRUE if the object was successfully destroyed, FALSE otherwise

Remarks

Destroys all the low-level Sapera resources needed by the counter object

SapCounter | SapCounter Class Members | Basic Class Hierarchy Chart

SapCounter::GetLocation

SapLocation **GetLocation**();

Remarks

Returns the location where the counter resource is located, as specified in the SapCounter constructor.

SapCounter | SapCounter Class Members | Basic Class Hierarchy Chart

SapCounter::SetCallbackInfo

virtual BOOL **SetCallbackInfo**(SapCounterCallback *pCallback*, void **pContext* = NULL);

Remarks

Sets the application callback method for counter events and the associated context. You can only call SetCallbackInfo before the Create method. See the SapCounter constructor for more details.

SapCounter | SapCounter Class Members | Basic Class Hierarchy Chart

SapCounter::GetCallback

SapCounterCallback **GetCallback**();

Remarks

Gets the current application callback method for counter events. The initial value for this attribute is NULL, unless you specify another value in the constructor.

See the SapCounter constructor for more details.

SapCounter | SapCounter Class Members | Basic Class Hierarchy Chart

SapCounter::GetContext

void ***GetContext**();

Remarks

Gets the application context associated with counter events. The initial value for this attribute is NULL, unless you specify another value in the constructor.

See the SapCounter constructor for more details.

SapCounter | SapCounter Class Members | Basic Class Hierarchy Chart

SapCounter::GetBaseUnits, SapCounter::SetBaseUnits

SapCounter::BaseUnits **GetBaseUnits**();

virtual BOOL **SetBaseUnits**(SapCounter::BaseUnits *baseUnits*);

Parameters

baseUnits	New basic counter units that may be based on any of the following:	
	SapCounter::UnitsTime	An internal hardware timer
	SapCounter::UnitsHSync	The horizontal sync signal from the acquisition device
	SapCounter::UnitsVSync	The vertical sync signal from the acquisition device
	SapCounter::UnitsExternal	A user-supplied external signal

Remarks

Gets/sets the basic counter units. The initial value for this attribute is UnitsTime.

You can only call SetBaseUnits before the Create method.

SapCounter | SapCounter Class Members | Basic Class Hierarchy Chart

SapCounter::GetDetectType, SapCounter::SetDetectType

SapCounter::DetectType **GetDetectType**();

virtual BOOL **SetDetectType**(SapCounter::DetectType *detectType*);

Parameters

detectType	New detection method where one of the following causes the counter value to change:	
	SapCounter::DetectRisingEdge	Rising edge of event triggering the counter
	SapCounter::DetectFallingEdge	Falling edge of event triggering the counter

Remarks

Gets/sets the detection method of events at which the counter value changes. The initial value for this attribute is DetectRisingEdge.

You can only call SetDetectType before the Create method.

SapCounter | SapCounter Class Members | Basic Class Hierarchy Chart

SapCounter::GetDirection, SapCounter::SetDirection

SapCounter::Direction **GetDirection**();

virtual BOOL **SetDirection**(SapCounter::Direction *direction*);

Parameters

direction	New counting direction. Can be one of the following values:	
	SapCounter::DirectionUp	Counter increments at each triggering event
	SapCounter::DirectionDown	Counter decrements at each triggering event

Remarks

Gets/sets the counting direction. The initial value for this attribute is DirectionUp.

You can only call SetDirection before the Create method.

SapCounter | SapCounter Class Members | Basic Class Hierarchy Chart

SapCounter::GetCount

virtual BOOL **GetCount**(int **pCount*);

Parameters

pCount	Pointer to counter value

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the current counter value with units being those returned by the GetBaseUnits method. You can only call GetCount after the Create method.

SapCounter | SapCounter Class Members | Basic Class Hierarchy Chart

SapCounter::GetHandle

virtual CORHANDLE **GetHandle**();

Remarks

Gets the low-level Sapera handle of the counter resource, which you may then use from the low-level Sapera functionality. The handle is only valid after you call the Create method.

See the *Sapera LT Basic Modules Reference Manual* for details on low-level Sapera functionality.

SapCounter | SapCounter Class Members | Basic Class Hierarchy Chart

SapCounter::Start

virtual BOOL **Start**();

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Starts the counter device. This is only relevant when the counter base units are not based on an internal hardware timer.

SapCounter | SapCounter Class Members | Basic Class Hierarchy Chart

SapCounter::Stop

virtual BOOL **Stop**();

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Stops the counter device. This is only relevant when the counter base units are not based on an internal hardware timer.

SapCounter | SapCounter Class Members | Basic Class Hierarchy Chart

SapCounter::EnableCallback

virtual BOOL **EnableCallback**(SapCounter::EventType *eventType*, int *elapsedTime*);

Parameters

eventType	Type of event that initiates calls to the application callback function. Only one value is currently supported:
	SapCounter::EventElapsedTime Events happen at regular time intervals
elapsedTime	Number of microseconds between each event

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Allows an application callback function to be called at specific counter events. See the SapCounter constructor for details about the application callback function.

SapCounter | SapCounter Class Members | Basic Class Hierarchy Chart

SapCounter::DisableCallback

virtual BOOL **DisableCallback**();

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Disables calls to the application callback function. See the SapCounter constructor and the EnableCallback method for more details.

SapCounter | SapCounter Class Members | Basic Class Hierarchy Chart

SapCounter::GetCapability

virtual BOOL **GetCapability**(int *cap*, void **pValue*);

Parameters

param	Low-level Sapera C library capability to read
pValue	Pointer to capability value to read back

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

This method allows direct read access to low-level Sapera C library capabilities for the counter module. It needs a pointer to a memory area large enough to receive the capability value, which is usually a 32-bit integer.

You will rarely need to use GetCapability. The SapCounter class already uses important capabilities internally for self-configuration and validation.

See the *Sapera LT Basic Modules Reference Manual* for a description of all capabilities and their possible values.

SapCounter | SapCounter Class Members | Basic Class Hierarchy Chart

SapCounter::GetParameter, SapCounter::SetParameter

virtual BOOL **GetParameter**(int *param*, void **pValue*);

virtual BOOL **SetParameter**(int *param*, int *value*);

virtual BOOL **SetParameter**(int *param*, void **pValue*);

Parameters

param	Low-level Sapera C library parameter to read or write
pValue	Pointer to parameter value to read back or to write
value	New parameter value to write

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

These methods allow direct read/write access to low-level Sapera C library parameters for the counter module. The GetParameter method needs a pointer to a memory area large enough to receive the parameter value, which is usually a 32-bit integer. The first form of SetParameter accepts a 32-bit value for the new value. The second form takes a pointer to the new value and is required when the parameter uses more than 32 bits of storage.

Note that you will rarely need to use these methods. You should first make certain that what you need is not already supported through the SapCounter Class. Also, directly setting parameter values may interfere with the correct operation of the class.

See the *Sapera LT Basic Modules Reference Manual* for a description of all parameters and their possible values.

SapCounter | SapCounter Class Members | Basic Class Hierarchy Chart

SapCounterCallbackInfo

The SapCounterCallbackInfo Class acts as a container for storing all arguments to the callback function for the SapCounter Class.

#include <SapClassBasic.h>

SapCounterCallbackInfo Class Members | Basic Class Hierarchy Chart

SapCounterCallbackInfo Class Members

Construction	
SapCounterCallbackInfo	Class constructor
Attributes	
GetCounter	Gets the SapCounter object associated with counter events
GetEventType	Gets the counter events that triggered the call to the application callback
GetEventCount	Gets the current count of counter events
GetContext	Gets the application context associated with counter events

SapCounterCallbackInfo | Basic Class Hierarchy Chart

Member Functions

The following functions are members of the SapCounterCallbackInfo Class.

SapCounterCallbackInfo::SapCounterCallbackInfo

```
SapCounterCallbackInfo(
  SapCounter *pCounter,
  void *pContext,
  SapCounter::EventType eventType,
  int eventCount
);
```

Parameters

pCounter	SapCounter object that calls the callback function
pContext	Pointer to the application context
eventType	Combination of counter events. See SapCounter::EnableCallback for a list a possible values
eventCount	Current counter event count

Remarks

SapCounter objects create an instance of this class before each call to the counter callback method, in order to combine all function arguments into one container.

SapCounter uses this class for reporting of counter events. The *pContext* parameter takes the value specified in the SapCounter Class constructor, *eventType* identifies the combination of events that triggered the call to the callback function, and *eventCount* increments by one at each call, starting at 1.

SapCounterCallbackInfo | SapCounterCallbackInfo Class Members | Basic Class Hierarchy Chart

SapCounterCallbackInfo::GetCounter

```
SapCounter *GetCounter();
```

Remarks

Gets the SapCounter object associated with counter events. See the SapCounter constructor for more details.

SapCounterCallbackInfo | SapCounterCallbackInfo Class Members | Basic Class Hierarchy Chart

SapCounterCallbackInfo::GetEventType

SapCounter::EventType **GetEventType**();

Remarks

Gets the combination of counter events that triggered the call to the application callback. See the SapCounter constructor for the list of possible values.

SapCounterCallbackInfo | SapCounterCallbackInfo Class Members | Basic Class Hierarchy Chart

SapCounterCallbackInfo::GetEventCount

int **GetEventCount**();

Remarks

Gets the current count of counter events. The initial value is 1 and increments after every call to the counter callback function.

SapCounterCallbackInfo | SapCounterCallbackInfo Class Members | Basic Class Hierarchy Chart

SapCounterCallbackInfo::GetContext

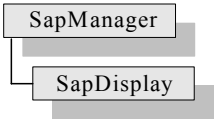
void ***GetContext**();

Remarks

Gets the application context associated with counter events. See the SapCounter constructor for more details.

SapCounterCallbackInfo | SapCounterCallbackInfo Class Members | Basic Class Hierarchy Chart

SapDisplay



The SapDisplay Class includes functionality to manipulate a display resource. There is at least one such resource for each display adapter (VGA board) in the system.

Note that SapView objects automatically manage an internal SapDisplay object for the default display resource. However, you must explicitly manage the object yourself if you need a display resource other than the default one.

#include <SapClassBasic.h>

SapDisplay Class Members | Base Class: SapManager | Basic Class Hierarchy Chart

SapDisplay Class Members

Construction	
SapDisplay	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Attributes	
GetLocation	Returns the location where the display resource is located
GetWidth	Gets the width (in pixels) for the current display mode
GetHeight	Gets the height (in lines) for the current display mode
GetPixelDepth	Gets the number of significant bits per pixel for the current display mode
GetRefreshRate	Gets the refresh rate for the current display mode
IsInterlaced	Verifies if the current display mode is interlaced or progressive
GetType	Gets the type of the display (primary or secondary)
IsPrimaryVGABoard	Verifies if the current display belongs to the primary VGA board in the system
IsOffscreenAvailable	Checks for offscreen display support of a specific buffer format

IsOverlayAvailable	Checks for overlay display support of a specific buffer format
GetHandle	Gets the low-level Sapera handle of the display resource
Operations	
GetDC	Gets the Windows Device Context corresponding to the entire screen
ReleaseDC	Releases the Windows Device Context corresponding to the entire screen
SetMode	Switches the display to a new mode (width, height, ...)
IsCapabilityValid	Checks for the availability of a low-level Sapera C library capability
IsParameterValid	Checks for the availability of a low-level Sapera C library parameter
GetCapability	Gets the value of a low-level Sapera C library capability
GetParameter SetParameter	Gets/sets the value of a low-level Sapera C library parameter

SapDisplay | Base Class: SapManager | Basic Class Hierarchy Chart

Member Functions

The following functions are members of the SapDisplay Class.

SapDisplay::SapDisplay

SapDisplay(
  SapLocation *loc* = SapLocation::ServerSystem
);

Parameters

loc SapLocation object specifying the server where the display resource is located and the index of the resource on this server

Remarks

The SapDisplay constructor does not actually create the low-level Sapera resources. To do this, you must call the Create method.

Note that SapView objects automatically manages an internal SapDisplay object for the default display resource; however, you must explicitly manage the object if you need a display resource other than the default one.

SapDisplay | SapDisplay Class Members | Basic Class Hierarchy Chart

SapDisplay::Create

virtual BOOL **Create**();

Return Value

Returns TRUE if the object was successfully created, FALSE otherwise

Remarks

Creates all the low-level Sapera resources needed by the display object.

If you allow a SapView object to automatically manage a SapDisplay object, then you do not need to call this method; otherwise, you must always call it before the SapView::Create method.

SapDisplay | SapDisplay Class Members | Basic Class Hierarchy Chart

SapDisplay::Destroy

virtual BOOL **Destroy**();

Return Value

Returns TRUE if the object was successfully destroyed, FALSE otherwise

Remarks

Destroys all the low-level Sapera resources needed by the display object.

If you allow a SapView object to automatically manage a SapDisplay object, then you do not need to call this method; otherwise, you must always call it after the SapView::Destroy method.

SapDisplay | SapDisplay Class Members | Basic Class Hierarchy Chart

SapDisplay::GetLocation

SapLocation **GetLocation**();

Remarks

Returns the location where the display resource is located. This usually corresponds to the system server, unless a specific server was specified in the SapDisplay constructor.

SapDisplay | SapDisplay Class Members | Basic Class Hierarchy Chart

SapDisplay::GetWidth

int **GetWidth**();

Remarks

Gets the width (in pixels) for the current display mode. This attribute has a value of value of 0 until the Create method is called.

SapDisplay | SapDisplay Class Members | Basic Class Hierarchy Chart

SapDisplay::GetHeight

int **GetHeight**();

Remarks

Gets the height (in lines) for the current display mode. This attribute has a value of value of 0 until the Create method is called.

SapDisplay | SapDisplay Class Members | Basic Class Hierarchy Chart

SapDisplay::GetPixelDepth

int **GetPixelDepth**();

Remarks

Gets the number of significant bits per pixel for the current display mode. This attribute has a value of value of 0 until the Create method is called.

SapDisplay | SapDisplay Class Members | Basic Class Hierarchy Chart

SapDisplay::GetRefreshRate

int **GetRefreshRate**();

Remarks

Gets the refresh rate (in Hz) for the current display mode. This attribute has a value of value of 0 until the Create method is called.

SapDisplay | SapDisplay Class Members | Basic Class Hierarchy Chart

SapDisplay::IsInterlaced

BOOL **IsInterlaced**();

Remarks

Returns TRUE if the current display mode is interlaced, FALSE if progressive (non-interlaced) . This attribute has a value of value of FALSE until the Create method is called.

SapDisplay | SapDisplay Class Members | Basic Class Hierarchy Chart

SapDisplay::GetType

SapDisplay::Type **GetType**();

Return Value

Display type, which can be one of the following values:

SapDisplay::TypeUnknown	Undetermined display type
SapDisplay::TypeSystem	A display under the control of the primary Windows display driver. It normally displays the Windows Desktop.
SapDisplay::TypeDuplicate	A secondary display that shows the same contents as the primary Windows VGA display
SapDisplay::TypeExtended	A secondary display that extends the desktop from the primary Windows VGA display
SapDisplay::TypeIndependent	A secondary display that is completely independent from the primary Windows VGA display

Remarks

Gets the type of the display (primary or secondary) . This attribute has a value of value of TypeUnknown until the Create method is called.

SapDisplay | SapDisplay Class Members | Basic Class Hierarchy Chart

SapDisplay::IsPrimaryVGABoard

virtual BOOL **IsPrimaryVGABoard**();

Remarks

Returns TRUE if the current display belongs to the primary VGA board in the system, FALSE otherwise. You can only call IsPrimaryVGABoard after the Create method.

SapDisplay | SapDisplay Class Members | Basic Class Hierarchy Chart

SapDisplay::IsOffscreenAvailable

virtual BOOL **IsOffscreenAvailable**(SapFormat *format*);

Remarks

Returns TRUE if offscreen display support is available for a given buffer format, FALSE otherwise. See the SapBuffer constructor for a list of possible values for *format.*

You can only call IsOffscreenAvailable after the Create method.

SapDisplay | SapDisplay Class Members | Basic Class Hierarchy Chart

SapDisplay::IsOverlayAvailable

virtual BOOL **IsOverlayAvailable**(SapFormat *format*);

Remarks

Returns TRUE if overlay display support is available for a given buffer format, FALSE otherwise. See the SapBuffer constructor for a list of possible values for *format.*

You can only call IsOverlayAvailable after the Create method.

SapDisplay | SapDisplay Class Members | Basic Class Hierarchy Chart

SapDisplay::GetHandle

virtual CORHANDLE **GetHandle**();

Remarks

Gets the low-level Sapera handle of the display resource, which you may then use from the low-level Sapera functionality. The handle is only valid after you call the Create method.

See the *Sapera LT Basic Modules Reference Manual* for details on low-level Sapera functionality.

SapDisplay | SapDisplay Class Members | Basic Class Hierarchy Chart

SapDisplay::GetDC

virtual BOOL **GetDC**(HDC **pDC*);

Parameters

pDC Pointer to display context value

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the Windows Device Context corresponding to the entire screen for the current display object.

SapDisplay | SapDisplay Class Members | Basic Class Hierarchy Chart

SapDisplay::ReleaseDC

virtual BOOL **ReleaseDC**();

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Releases the Windows Device Context corresponding to the entire screen for the current display object.

SapDisplay | SapDisplay Class Members | Basic Class Hierarchy Chart

SapDisplay::SetMode

virtual BOOL **SetMode**(int *width*, int *height*, SapFormat *format*, int *refresh*);

Parameters

width	Width (in pixels) for the new display mode	
height	Height (in lines) for the new display mode	
format	Data format for the new display mode, which can be one of the following:	
	SapFormatMono8	8-bit monochrome
	SapFormatRGB565	16-bit RGB (5 for red, 6 for green, 5 for blue)
	SapFormatRGB8888	32-bit RGB (8 for each of red/green/blue, 8 for alpha)
refresh	Refresh rate (in Hz) for the new display mode	

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Programs the current display to a new mode using the supplied parameters.

The system display (that normally shows the Windows Desktop) may not be reprogrammed. SetMode also fails if the specified mode is not supported by the display hardware.

You can only call this method before the Create method.

SapDisplay | SapDisplay Class Members | Basic Class Hierarchy Chart

SapDisplay::IsCapabilityValid

virtual BOOL **IsCapabilityValid**(int *cap*);

Parameters

cap Low-level Sapera C library capability to check

Return Value

Returns TRUE if the capability is supported, FALSE otherwise

Remarks

Checks for the availability of a low-level Sapera C library capability for the display module. Call this method before GetCapability to avoid invalid or not available capability errors.

IsCapabilityValid is rarely needed. The SapDisplay class already uses important capabilities internally for self-configuration and validation.

See the *Sapera LT Basic Modules Reference Manual* for a description of all capabilities and their possible values.

SapDisplay | SapDisplay Class Members | Basic Class Hierarchy Chart

SapDisplay::IsParameterValid

virtual BOOL **IsParameterValid**(int *param*);

Parameters

param Low-level Sapera C library parameter to check

Return Value

Returns TRUE if the parameter is supported, FALSE otherwise

Remarks

Checks for the availability of a low-level Sapera C library parameter for the display module. Call this method before GetParameter to avoid invalid or not available parameter errors.

IsParameterValid is rarely needed. The SapDisplay class already uses important parameters internally for self-configuration and validation.

See the *Sapera LT Basic Modules Reference Manual* for a description of all parameters and their possible values.

SapDisplay | SapDisplay Class Members | Basic Class Hierarchy Chart

SapDisplay::GetCapability

virtual BOOL **GetCapability**(int *cap*, void **pValue*);

Parameters

Param	Low-level Sapera C library capability to read
PValue	Pointer to capability value to read back

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

This method allows direct read access to low-level Sapera C library capabilities for the display module. It needs a pointer to a memory area large enough to receive the capability value, which is usually a 32-bit integer.

You will rarely need to use GetCapability. The SapDisplay Class already uses important capabilities internally for self-configuration and validation.

See the *Sapera LT Basic Modules Reference Manual* for a description of all capabilities and their possible values.

SapDisplay | SapDisplay Class Members | Basic Class Hierarchy Chart

SapDisplay::GetParameter, SapDisplay::SetParameter

virtual BOOL **GetParameter**(int *param*, void **pValue*);

virtual BOOL **SetParameter**(int *param*, int *value*);

virtual BOOL **SetParameter**(int *param*, void **pValue*);

Parameters

param	Low-level Sapera C library parameter to read or write
pValue	Pointer to parameter value to read back or to write
value	New parameter value to write

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

These methods allow direct read/write access to low-level Sapera C library parameters for the display module. The GetParameter method needs a pointer to a memory area large enough to receive the parameter value, which is usually a 32-bit integer. The first form of SetParameter accepts a 32-bit value for the new value. The second form takes a pointer to the new value, and is required when the parameter uses more than 32-bits of storage.

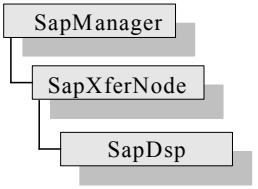
Note that you will rarely need to use these methods. You should first make certain that what you need is not

already supported through the SapDisplay Class. Also, directly setting parameter values may interfere with the correct operation of the class.

See the *Sapera LT Basic Modules Reference Manual* for a description of all parameters and their possible values.

SapDisplay | SapDisplay Class Members | Basic Class Hierarchy Chart

SapDsp



The SapDsp Class includes the functionality to manipulate a DSP resource. It is used as an intermediate node to modify data transferred data from an acquisition resource to another transfer node (such as buffer or CAB).

#include <SapClassBasic.h>

SapDsp Class Members | Base Class: SapManager | Basic Class Hierarchy Chart

SapDsp Class Members

Construction	
SapDsp	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Attributes	
GetFile SetFile	Gets/sets the name of the program file to load into the DSP memory

SapDsp | Base Class: SapManager | Basic Class Hierarchy Chart

Member Functions

The following functions are members of the SapDsp Class.

SapDsp::SapDsp

SapDsp(
  SapLocation *loc* = SapLocation::ServerSystem,
  const char **fileName* = "",
  SapXferNode **pSrcNode* = NULL
);

SapDsp(
  SapLocation *loc*,
  const char **fileName*,
  SapXferParams *xferParams*
);

Parameters

loc	SapLocation object specifying the server where the DSP resource is located and the index of the resource on this server
fileName	Name of the program file to load into the DSP memory
pSrcNode	Source node object. To ensure transfer compatibility, this object must match the source node specified when adding a transfer pair (SapXferPair) to the SapTransfer object.
xferParams	Transfer parameters of the source node. To ensure transfer compatibility, its attributes must match the equivalent attributes of the source node specified when adding a transfer pair (SapXferPair) to the SapTransfer object.

Remarks

The SapDsp constructor does not actually create the low-level Sapera resources. To do this, you must call the Create method.

Use the SapDsp object only for loading the DSP program file into memory. To transfer data through the DSP device, you must use the SapTransfer Class (or one of its derived classes) and specify the SapDsp object as a parameter. The data transfer is then controlled by the SapTransfer Class.

SapDsp | SapDsp Class Members | Basic Class Hierarchy Chart

SapDsp::Create

virtual BOOL **Create**();

Return Value

Returns TRUE if the object was successfully created, FALSE otherwise

Remarks

Creates all the low-level Sapera resources needed by the DSP object and loads the DSP file into memory. Always call this method before SapTransfer::Create.

SapDsp | SapDsp Class Members | Basic Class Hierarchy Chart

SapDsp::Destroy

virtual BOOL **Destroy**();

Return Value

Returns TRUE if the object was successfully destroyed, FALSE otherwise

Remarks

Destroys all the low-level Sapera resources needed by the DSP object. Always call this method after SapTransfer::Destroy.

SapDsp | SapDsp Class Members | Basic Class Hierarchy Chart

SapDsp::GetFile, SapDsp::SetFile

const char ***GetFile**();

virtual BOOL **SetFile**(const char **fileName*);

Remarks

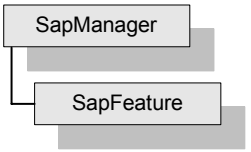
Gets/sets the name of the program file to load into DSP memory.

You usually set the initial value for this attribute in the SapDsp constructor. If you do not specify a value at that time, then it defaults to an empty string.

You can only call SetFile before the Create method.

SapDsp | SapDsp Class Members | Basic Class Hierarchy Chart

SapFeature



The purpose of the *SapFeature* class is to retrieve the feature information from the *SapAcqDevice* class. Each feature supported by the *SapAcqDevice* class provides a set of capabilities such as name, type, access mode, and so forth, that can be obtained through the *SapFeature* class. The *SapFeature* class is used by the *SapAcqDevice::GetFeatureInfo* function.

#include <SapClassBasic.h>

SapFeature Class Members | Basic Class Hierarchy Chart

SapFeature Class Members

Construction	
SapFeature	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Attributes	
GetLocation SetLocation	Gets/sets the location where the feature resource is located
GetHandle	Gets the low-level Sapera handle of the feature resource
General Parameters	
GetName	Returns the name of the feature
GetType	Returns the type of the feature
IsStandard	Returns whether a feature is standard or custom
GetAccessMode	Returns the type of access for a feature
GetPollingTime	Returns the interval of time between two consecutive feature updates
GetToolTip	Returns a small text representing the explanation of the feature

GetDisplayName	Returns the name of the feature in a more descriptive way than *GetName*
GetRepresentation	Returns the mathematical representation of a integer or float feature
GetSign	Returns whether an integer/float feature is signed or not
GetSiUnit	Returns the physical unit representing the feature in the international system (SI)
GetCategory	Returns the category of features the current feature belongs to
GetWriteMode	Returns whether or not a feature can be modified when the transfer object is connected and/or acquiring
IsSavedToConfigFile SetSavedToConfigFile	Gets/sets whether or not a feature is saved to the configuration file when calling *SaveFeatures*
GetSiToNativeExp10	Returns the feature conversion factor from SI unit to native unit
GetVisibility	Returns the level of visibility assigned to a feature
Integer/float-Parameters	
GetMin	Returns the minimum acceptable value for a feature
GetMax	Returns the maximum acceptable value for a feature
GetInc	Returns the minimum acceptable increment for an integer or a float feature
Enumeration-Parameters	
GetEnumCount	Returns the number of items in an enumeration
GetEnumString	Returns the enumeration string corresponding to a specified index
GetEnumValue	Returns the enumeration value corresponding to a specified index
IsEnumEnabled	Returns whether or not the enumeration item corresponding to a specified index is enabled
GetEnumStringFromValue	Returns the enumeration string corresponding to a specified enumeration value
GetEnumValueFromString	Returns the enumeration value corresponding to a specified enumeration string

SapFeature | Basic Class Hierarchy Chart

SapFeature::SapFeature

SapFeature(SapLocation *location* = SapLocation::ServerSystem);

Parameters

location SapLocation object specifying where the feature is located. The location must be the same as that of the *SapAcqDevice* object

Remarks

The *SapFeature* constructor does not actually create the low-level Sapera resources. To do this, you must call the SapFeature::Create method. Upon creation the feature object is reset with null values. To fill-in a feature object, call the SapAcqDevice::GetFeatureInfo function.

SapFeature | SapFeature Class Members | Basic Class Hierarchy Chart

SapFeature::Create

BOOL **Create**();

Remarks

Creates all the low-level Sapera resources needed by the feature object. Call this method before passing the object to *SapAcqDevice::GetFeatureInfo*.

Return Value

Returns TRUE if the object was successfully created, FALSE otherwise

SapFeature | SapFeature Class Members | Basic Class Hierarchy Chart

SapFeature::Destroy

BOOL **Destroy**();

Remarks

Destroys all the low-level Sapera resources needed by the feature object.

Return Value

Returns TRUE if the object was successfully destroyed, FALSE otherwise

SapFeature | SapFeature Class Members | Basic Class Hierarchy Chart

SapFeature::GetAccessMode

BOOL **GetAccessMode**(AccessMode **accessMode*);

Parameters

accessMode

SapFeature::AccessUndefined
Undefined access mode

SapFeature::AccessRW
The feature can be read and written. Most of the features are of this type.

SapFeature::AccessRO
The feature can only be read. Certain type of features such as “Sensor Temperature” and “Sensor Resolution” cannot be written.

SapFeature::AccessWO
The feature can only be written. Some features represent commands (or actions) such as “starting a calibration algorithm” for example. This kind of feature cannot be read back.

SapFeature::AccessNP
The feature is not present. The feature is visible in the interface but is not implemented for this device.

SapFeature::AccessNE
The feature is present but not enabled. Often used when a feature depends on another feature’s value.

Remarks

Return the type of access for a feature. See the value descriptions above.

SapFeature | SapFeature Class Members | Basic Class Hierarchy Chart

SapFeature::GetCategory

BOOL **GetCategory**(char **category*, int *categorySize*);

Parameters

category	Address of a buffer to return the string. Must be at least 64 charaters long
categorySize	Size of the buffer pointed to by *category* (in bytes)

Remarks

Returns the category of features the current feature belongs to. All the features are divided into categories to simplify the presentation of features coming from a large feature set. The categories are useful to present a categorized list of features in a graphical user interface.

SapFeature | SapFeature Class Members | Basic Class Hierarchy Chart

SapFeature::GetDisplayName

BOOL **GetDisplayName**(char **displayName*, int *displayNameSize*);

Parameters

displayName	Address of a buffer to return the string. Must be at least 64 charaters long
displayNameSize	Size of the buffer pointed to by *displayName* (in bytes)

Remarks

Returns the name of the feature in a more descriptive way than *GetName*. This name can be used for listing features in a graphical user interface.

SapFeature | SapFeature Class Members | Basic Class Hierarchy Chart

SapFeature::GetEnumCount

BOOL **GetEnumCount**(int **enumCount*);

Parameters

EnumCount Returns the number of items in an enumeration

Remarks

Returns the number of items in an enumeration. Use this function along with *GetEnumString* and *GetEnumValue* to enumerate all the items contained within an enumeration feature.

SapFeature | SapFeature Class Members | Basic Class Hierarchy Chart

SapFeature::GetEnumString

BOOL **GetEnumString**(int *enumIndex*, char **enumString*, int *enumStringSize*);

Parameters

enumIndex Index of the enumeration item. Ranges from 0 to *GetEnumCount* - 1

enumString Address of a buffer to return the string

enumStringSize Size of the buffer pointed to by *enumString* (in bytes)

Remarks

Returns the enumeration string corresponding to a specified index. Use this function along with *GetEnumCount* and *GetEnumValue* to enumerate all the items contained within an enumeration feature.

SapFeature | SapFeature Class Members | Basic Class Hierarchy Chart

SapFeature::GetEnumStringFromValue

BOOL **GetEnumStringFromValue**(int *enumValue*, char **enumString*, int *enumStringSize*);

Parameters

enumValue Value to look for in the enumeration items

enumString Address of a buffer to return the string

enumStringSize Size of the buffer pointed to by *enumString* (in bytes)

Remarks

Returns the enumeration string corresponding to a specified enumeration value. For example you may use this function to retrieve the string corresponding to an enumeration value returned by the *SapAcqDevice::GetFeatureValue* function.

SapFeature | SapFeature Class Members | Basic Class Hierarchy Chart

SapFeature::GetEnumValue

BOOL **GetEnumValue**(int *enumIndex*, int **enumValue*);

Parameters

enumIndex	Index of the enumeration item. Ranges from 0 to *GetEnumCount* - 1
enumValue	Returns the value associated with the item specified by *enumIndex*

Remarks

Returns the enumeration value corresponding to a specified index. Use this function along with *GetEnumCount* and *GetEnumString* to enumerate all the items contained within an enumeration feature.

SapFeature | SapFeature Class Members | Basic Class Hierarchy Chart

SapFeature::GetEnumValueFromString

BOOL **GetEnumValueFromString**(const char **enumString*, int **enumValue*);

Parameters

enumString	String to look for in the enumeration items
enumValue	Returns the value associated with the specified string

Remarks

Returns the enumeration value corresponding to a specified enumeration string. For example you may use this function to retrieve the value corresponding to a known enumeration string and then set this value to the device using the *SapAcqDevice::SetFeatureValue* function.

SapFeature | SapFeature Class Members | Basic Class Hierarchy Chart

SapFeature::GetHandle

CORHANDLE **GetHandle**() const;

Remarks

Gets the low-level Sapera handle of the feature resource. This function should not be used unless a handle to the low-level C API is required to access some functionality not exposed in the C++ API.

SapFeature | SapFeature Class Members | Basic Class Hierarchy Chart

SapFeature::GetInc

BOOL **GetInc**(UINT32 **incValue*);
BOOL **GetInc**(UINT64 **incValue*);
BOOL **GetInc**(float **incValue*);
BOOL **GetInc**(double **incValue*);

Parameters

incValue Returns the increment value

Remarks

Returns the minimum acceptable increment for an integer or a float feature. Some features cannot vary by increments of 1. Their value must be a multiple of a certain increment. For example the buffer cropping dimensions might require to be a multiple of 4 in order to optimize the data transfer.

SapFeature | SapFeature Class Members | Basic Class Hierarchy Chart

SapFeature::GetLocation, SapFeature::SetLocation

SapLocation **GetLocation**() const;
BOOL **SetLocation**(SapLocation *location*);

Parameters

location Specifies where the feature resource is located

Remarks

Gets/sets the location of the feature resource. The location must be the same as that of the *SapAcqDevice* object. You can only call *SetLocation* before the *Create* method.

SapFeature | SapFeature Class Members | Basic Class Hierarchy Chart

SapFeature::GetMax

BOOL **GetMax**(UINT32 **maxValue*);
BOOL **GetMax**(UINT64 **maxValue*);
BOOL **GetMax**(float **maxValue*);
BOOL **GetMax**(double **maxValue*);

Parameters

maxValue Returns the maximum value

Remarks

Returns the maximum acceptable value for a feature. For integer and float types use the overload corresponding to the type of the feature. For a string type the maximum length of the string (excluding the trailing null character) is returned by the UINT32 overload.

SapFeature | SapFeature Class Members | Basic Class Hierarchy Chart

SapFeature::GetMin

BOOL **GetMin**(UINT32 **minValue*);
BOOL **GetMin**(UINT64 **minValue*);
BOOL **GetMin**(float **minValue*);
BOOL **GetMin**(double **minValue*);

Parameters

minValue Returns the minimum value

Remarks

Returns the minimum acceptable value for a feature. For integer and float types use the overload corresponding to the type of the feature. For a string type the minimum length of the string (excluding the trailing null character) is returned by the UINT32 overload.

SapFeature | SapFeature Class Members | Basic Class Hierarchy Chart

SapFeature::GetName

BOOL **GetName**(char ***name**, int *nameSize*);

Parameters

name	Address of a buffer to return the string. Must be at least 64 characters long
nameSize	Size of the buffer pointed to by *name* (in bytes)

Remarks

Returns the name of the feature. The name of a feature can be used as an index to the feature set in the *SapAcqDevice* class. This string should not be used for display in a graphical user interface. Instead the *GetDisplayName* function provides a more descriptive name.

SapFeature | SapFeature Class Members | Basic Class Hierarchy Chart

SapFeature::GetPollingTime

BOOL **GetPollingTime**(int ***pollingTime**);

Parameters

pollingTime	Returns the polling time, in ms.

Remarks

Returns the interval of time between two consecutive feature updates. Some read-only features such as “Sensor Temperature” for instance must be read from the device at a certain frequency in order to refresh to feature module. You can modify this value to either increase or decrease the reshresh rate.

SapFeature | SapFeature Class Members | Basic Class Hierarchy Chart

SapFeature::GetRepresentation

BOOL **GetRepresentation**(Representation ***representation**);

Parameters

representation

SapFeature::RepresentationUndefined
Undefined representation

SapFeature::Representation Linear
The feature follows a linear scale

SapFeature::Representation Logarithmic
The feature follows a logarithmic scale

SapFeature::RepresentationBoolean
The feature is a boolean (can have two values: zero or non-zero)

Remarks

Returns the mathematical representation of a integer or float feature. See possible values above.

SapFeature | SapFeature Class Members | Basic Class Hierarchy Chart

SapFeature::GetSign

BOOL **GetSign**(Sign **sign*);

Parameters

sign

SapFeature::SignUndefined
Sign is undefined

SapFeature::Signed
The feature is a signed integer of float

SapFeature::Unsigned
The feature is an unsigned integer of float

Remarks

Returns the sign of an integer or float feature. This information is useful when reading and writing feature values. By knowing the sign of the feature value you can “type cast” it to the corresponding C/C++ type. See possible values above.

SapFeature | SapFeature Class Members | Basic Class Hierarchy Chart

SapFeature::GetSiToNativeExp10

BOOL **GetSiToNativeExp10**(int **exponent*);

Parameters

exponent The value is the exponent of a base 10. It can be negative or positive.

Remarks

Returns the exponent of the base 10 factor used to convert the value of a feature from SI unit (international system) to native unit (the unit used to read/write the feature through the API). The following equation describes the relation between the two unit systems:

$$V_{NATIVE} = V_{SI} * 10^{E}$$

Where *V* is the value of a feature and *E* is the current parameter.

Example 1

You want to set the camera integration time to a known value in seconds. The "IntegrationDuration" feature is represented in microseconds. Therefore the current parameter value is 6. For instance if the desired integration time is 0.5 second you can compute the value to pass to *SapAcqDevice::SetFeatureValue* as follows:

$$V_{NATIVE} = 0.5 * 10^{6} = 500000$$

Example 2

You want to monitor the temperature of the camera sensor. The "SensorTemperature" feature is reported in tenths of degrees Celcius. Therefore the current parameter value is 1. For instance if the feature value returned by *SapAcqDevice::GetFeatureValue* is 205 then you can compute the temperature in Celcius as follows:

$$V_{SI} = \frac{V_{NATIVE}}{10^{E}} = \frac{205}{10^{1}} = 20.5$$

Use the *GetSiUnit* function to retrieve the SI unit corresponding to the feature to monitor.

SapFeature | SapFeature Class Members | Basic Class Hierarchy Chart

SapFeature::GetSiUnit

BOOL **GetSiUnit**(char **unit*, int *unitSize*);

Parameters

unit	Address of a buffer to return the string describing the unit. Must be at least 32 characters long
unitSize	Size of the buffer pointed to by *unit* (in bytes)

Remarks

Returns the physical unit representing the feature in the international system (SI). Examples of units are Volts, Pixels, Celsius Degrees, etc. This information is useful to present in a graphical user interface.

Most of the time the unit used by the feature (the native unit) is NOT the SI unit directly but a multiple of it, for example the exposure time may be represented in microseconds instead of seconds. To convert the feature value to the SI unit you must use the conversion factor provided by the *GetSiToNativeExp10* function.

SapFeature | SapFeature Class Members | Basic Class Hierarchy Chart

SapFeature::GetToolTip

BOOL **GetToolTip**(char **tooltip*, int *tooltipSize*);

Parameters

tooltip	Address of a buffer to return the string. Must be at least 64 characters long
tooltipSize	Size of the buffer pointed to by *tooltip* (in bytes)

Remarks

Returns a small text representing the explanation of the feature. This information can be used to implement tool tips in a graphical user interface.

SapFeature | SapFeature Class Members | Basic Class Hierarchy Chart

SapFeature::GetType

BOOL **GetType**(Type **type*);

Parameters

type

SapFeature::TypeUndefined
Undefined type

SapFeature::TypeInt32
32-bit integer

SapFeature::TypeInt64
64-bit integer

SapFeature::TypeFloat
32-bit floating-point

SapFeature::TypeDouble
64-bit floating-point

SapFeature::TypeBool
Boolean

SapFeature::TypeEnum
Enumeration

SapFeature::TypeString
ASCII character string

SapFeature::TypeBuffer
Buffer handle

SapFeature::TypeLut
LUT handle

Remarks

Return the type of a feature. See the value descriptions above.

SapFeature | SapFeature Class Members | Basic Class Hierarchy Chart

SapFeature::GetVisibility

BOOL **GetVisibility**(Visibility **visibility*);

Parameters

visibility

SapFeature::VisibilityUndefined
Undefined visibility level

SapFeature::VisibilityBeginner
Specifies that the feature should be made visible to any user

SapFeature::VisibilityExpert
Specifies that the feature should be made visible to users with a certain level of expertise

SapFeature::VisibilityGuru
Specifies that the feature should be made visible to users with a high level of expertise

SapFeature::VisibilityInvisible
Specifies that the feature should not be made visible to any user. This level of visibility is normally used on obsolete or internal features

Remarks

Returns the level of visibility assigned to a feature. This information is useful to classify the features in a graphical user interface in terms of user expertise. See the possible value above.

SapFeature | SapFeature Class Members | Basic Class Hierarchy Chart

SapFeature::GetWriteMode

BOOL **GetWriteMode**(WriteMode **writeMode*);

Parameters

writeMode

SapFeature::WriteUndefined
Undefined write mode

SapFeature::WriteAlways
The feature can always be written

SapFeature::WriteNotAcquiring
The feature can only be written when the transfer object is not acquiring. If the transfer is currently acquiring you must stop the acquisition using *SapTransfer::Freeze / SapTransfer::Wait* before modifying the feature value

SapFeature::WriteNotConnected
The feature can only be written when the transfer object is not connected. If the transfer is currently connected you must disconnect it using *SapTransfer::Disconnect* (or *SapTransfer::Destroy*) before modifying the feature value. After modifying the value

reconnect the transfer object using *SapTransfer::Connect* (or *SapTransfer::Create*)

Remarks

Returns whether or not a feature can be modified when the transfer module is connected and/or acquiring. Some features like the buffer dimensions for example cannot be changed while data is being transfered to the buffer. Use this information to prevent an application from changing certain features when the transfer object is connected and/or acquiring.

SapFeature | SapFeature Class Members | Basic Class Hierarchy Chart

SapFeature::IsEnumEnabled

BOOL **IsEnumEnabled**(int *enumIndex*, BOOL **enabled*);

Parameters

enumIndex	Index of the enumeration item. Ranges from 0 to *GetEnumCount* - 1
enabled	Returns TRUE if the item is enabled, FALSE otherwise.

Remarks

Returns whether or not the enumeration item corresponding to a specified index is enabled. Each item in an enumeration is present for all the application duration. However an enumeration item may be dynamically enabled/disabled according to the value of another feature. Use this function to determine the enable state of an item at a given time.

SapFeature | SapFeature Class Members | Basic Class Hierarchy Chart

SapFeature::IsSavedToConfigFile, SapFeature::SetSavedToConfigFile

BOOL **IsSavedToConfigFile**(BOOL **savedToConfigFile*);
BOOL **SetSavedToConfigFile**(BOOL *savedToConfigFile*);

Parameters

savedToConfigFile	TRUE for allowing the feature to be saved, FALSE otherwise.

Remarks

Specifies whether or not a feature is saved to the configuration file when calling *SapAcqDevice::SaveFeatures*. All the features are assigned a default behavior. For example the read-only features are not saved while the read-write features are. You can however change the default behavoir using this parameter. For example a read-only feature such as “Sensor Temperature” is not saved by default. You set this parameter to TRUE to force the feature to be written to the configuration file.

If you force read-only features to be saved those features will not be restored when loading back the CCF file. The reason is that the features are not writable to the device.

SapFeature | SapFeature Class Members | Basic Class Hierarchy Chart

SapFeature::IsStandard

BOOL **IsStandard**(BOOL **isStandard*);

Parameters

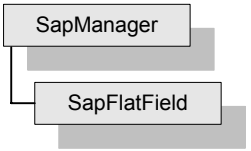
isStandard Returns TRUE if the feature is standard, FALSE otherwise.

Remarks

Returns whether a feature is standard or custom. Most of the features are standard. However sometimes custom features might be provided as part of a special version of an acquisition device driver.

SapFeature | SapFeature Class Members | Basic Class Hierarchy Chart

SapFlatField



The purpose of the SapFlatField Class is to support flat-field correction on monochrome images. The first scenario is where images are acquired from a camera. Flat-field correction is then performed either by the acquisition device (if supported) or through software. The second scenario is where images are taken from another source (for example, loaded from disk). Only the software implementation is then available.

#include <SapClassBasic.h>

SapFlatField Class Members | Base Class: SapManager | Basic Class Hierarchy Chart

SapFlatField Class Members

Construction	
SapFlatField	Class constructor
Create	Allocates the internal resources
Destroy	Releases the internal resources
Attributes	
GetNumFlatField SetNumFlatField	Gets/sets the number of buffer pairs for gain and offset coefficients
GetAcquisition, SetAcquisition	Gets/sets the acquisition object for acquiring images and flat-field correction
GetBuffer, SetBuffer	Gets/sets the buffer object for operating the flat-field correction without an acquisition object
GetBufferOffset, GetBufferGain	Gets the buffer objects for the flat-field correction gain and offset coefficients
IsEnabled	Checks if flat-field correction is enabled
IsPixelReplacement	Checks if replacement of defective pixels is enabled

IsSoftware	Checks if flat-field correction is performed in software or using the hardware
GetCorrectionType SetCorrectionType	Gets/sets line scan vs area scan correction type
GetVideoType SetVideoType	Gets/sets the acquisition video type (monochrome or Bayer)
GetDeviationMaxBlack SetDeviationMaxBlack	Gets/sets the maximum deviation of the calculated coefficients towards black
GetDeviationMaxWhite SetDeviationMaxWhite	Gets/sets the maximum deviation of the calculated coefficients towards white
GetGainDivisor, SetGainDivisor	Gets/sets the factor by which a gain coefficient has to be divided for getting a unitary scale factor.
SetOffsetMinMax	Sets the minimum and maximum values for computed offset values
SetGainMinMax	Sets the minimum and maximum values for computed gain values
GetNumLinesAverage, SetNumLinesAverage	Gets/sets the number of lines to be averaged, in the image used for doing the calibration before computing the gain and offset coefficients for linescan video source.
GetVerticalOffset SetVerticalOffset	Gets/sets the vertical line scan averaging offset in a full frame
GetIndex SetIndex	Gets/sets the index of the current buffer pair for gain and offset coefficients
Operations	
Next	Increments the index of the current buffer pair for gain and offset coefficients
Load	Loads gain and offset buffer data from disk files or from existing buffer objects
Save	Saves gain and offset buffer data to disk files
Clear	Clears the gain and offset buffers
ComputeOffset	Calculates the flat-field correction offset coefficients
ComputeGain	Calculates the flat-field correction gain coefficients
Enable	Enables/disables flat-field correction
EnablePixelReplacement	Enables/disables replacement of defective pixels
Execute	Performs the software implementation of flat-field correction
GetAverage	Gets average pixel value and standard deviation for a buffer
GetStats	Gets statistics for a buffer subtracted from the offset buffer

SapFlatField | Base Class: SapManager | Basic Class Hierarchy Chart

Member Functions

The following functions are members of the SapFlatField Class.

SapFlatField::SapFlatField

SapFlatField(
  SapAcquisition **pAcq*,
  int *numFlatField* = 1
);

SapFlatField(
  SapBuffer **pBuffer*,
  int *numFlatField* = 1
);

Parameters

pAcq	SapAcquisition object to be used for image acquisition and for flat-field correction (if available in hardware)
numFlatField	Number of gain/offset buffer resource pairs
pBuffer	SapBuffer object to be used to determine the width, height and format for the flat-field correction gain and offset buffer objects

Remarks

The SapFlatField constructor does not actually create the internal resources. To do this, you must call the Create method.

The *numFlatField* argument specifies the number of buffer pairs for gain and offset coefficients. This is useful when consecutive images from the acquisition device need different sets of gain/offset coefficients. When hardware correction is used, these buffer pairs are used together with the flat-field cycle mode value specified using the SapXferPair::GetFlatFieldCycleMode, SapXferPair::SetFlatFieldCycleMode method.

The constructor with a SapBuffer object is used only for offline operation (no acquisition device), so that only software correction will be available.

SapFlatField | SapFlatField Class Members | Basic Class Hierarchy Chart

SapFlatField::Create

virtual BOOL **Create**();

Return Value

Returns TRUE if the object was successfully created, FALSE otherwise

Remarks

Creates all the internal resources needed by the flat-field correction object

SapFlatField | SapFlatField Class Members | Basic Class Hierarchy Chart

SapFlatField::Destroy

virtual BOOL **Destroy**();

Return Value

Returns TRUE if the object was successfully destroyed, FALSE otherwise

Remarks

Destroys all the internal resources needed by the flat-field correction object

SapFlatField | SapFlatField Class Members | Basic Class Hierarchy Chart

SapFlatField::GetNumFlatField, SapFlatField::SetNumFlatField

int **GetNumFlatField**();

virtual BOOL **SetNumFlatField**(int *numFlatField*);

Remarks

Gets/sets the number of buffer pairs for gain and offset coefficients. This is useful when consecutive images from the acquisition device need different sets of gain/offset coefficients. When hardware correction is used, these buffer pairs are used together with the flat-field cycle mode value specified using the SapXferPair::GetFlatFieldCycleMode, SapXferPair::SetFlatFieldCycleMode method.

Unless you specify another value in the constructor the initial value for this attribute is 1.

You can only call SetNumFlatField before the Create method. Calling this method will also automatically reset the index of the current buffer pair for gain and offset coefficients to 0.

SapFlatField | SapFlatField Class Members | Basic Class Hierarchy Chart

SapFlatField::GetAcquisition, SapFlatField::SetAcquisition

SapAcquisition ***GetAcquisition**();

virtual BOOL **SetAcquisition**(SapAcquisition **pAcq*);

Remarks

Gets/sets the SapAcquisition object to be used for image acquisition and for flat-field correction. You can only call SetAcquisition before the Create method.

SapFlatField | SapFlatField Class Members | Basic Class Hierarchy Chart

SapFlatField::GetBuffer, SapFlatField::SetBuffer

SapBuffer ***GetBuffer**();

virtual BOOL **SetBuffer**(SapBuffer **pBuffer*);

Remarks

Gets/sets the SapBuffer object to be used to determine the width, height and format for the flat-field correction gain and offset buffer objects. You can only call SetBuffer before the Create method.

SapFlatField | SapFlatField Class Members | Basic Class Hierarchy Chart

SapFlatField::GetBufferOffset, SapFlatField::GetBufferGain

SapBuffer ***GetBufferOffset**();

SapBuffer ***GetBufferGain**();

Remarks

Gets the SapBuffer object that contains the flat-field correction gain and offset coefficients

SapFlatField | SapFlatField Class Members | Basic Class Hierarchy Chart

SapFlatField::IsEnabled

BOOL **IsEnabled**();

Remarks

Checks if flat-field correction is enabled. The initial value for this attribute depends on the acquisition device.

Use the Enable method if you need to explicitly enable or disable flat-field correction.

SapFlatField | SapFlatField Class Members | Basic Class Hierarchy Chart

SapFlatField::IsPixelReplacement

BOOL **IsPixelReplacement**();

Remarks

Checks if replacement of defective pixels is enabled.

Pixel replacement is used when calling the Execute method to perform the software implementation of flat-field correction. If it is TRUE, then defective pixel values are replaced by the value of a neighboring pixel. This is usually the one to the left of the current pixel, except for the first column, where the value of the pixel to the right is used instead.

The initial value for this attribute is TRUE.

Use the EnablePixelReplacement method if you need to explicitly enable or disable this feature.

SapFlatField | SapFlatField Class Members | Basic Class Hierarchy Chart

SapFlatField::IsSoftware

BOOL **IsSoftware**();

Remarks

Checks if flat-field correction is available using a software implementation only, or if it is present in the acquisition device.

SapFlatField | SapFlatField Class Members | Basic Class Hierarchy Chart

SapFlatField::GetCorrectionType, SapFlatField::SetCorrectionType

SapFlatField::CorrectionType **GetCorrectionType**();

virtual BOOL **SetCorrectionType**(SapFlatField::CorrectionType *correctionType*);

Parameters

correctionType	Flat-field correction type may be one of the following values	
	SapFlatField::CorrectionTypeField	Correction is performed on full frames
	SapFlatField::CorrectionTypeLine	Correction is performed on individual lines
	SapFlatField::CorrectionTypeInvalid	Invalid correction type

Remarks

Gets/sets the flat-field correction type.

The initial value for this attribute is SapFlatField::CorrectionTypeInvalid. It is then set according to the acquisition device scan type when calling the Create method.

SapFlatField | SapFlatField Class Members | Basic Class Hierarchy Chart

SapFlatField::GetVideoType, SapFlatField::SetVideoType

SapAcquisition::VideoType **GetVideoType**();

virtual BOOL **SetVideoType**(SapAcquisition::VideoType *videoType*);

Parameters

videoType New acquisition video type (SapAcquisition::VideoMono or SapAcquisition::VideoBayer)

Remarks

Gets/sets the acquisition video type.

The initial value for this attribute is monochrome. If the current flat-field object is associated with a SapAcquisition object (see the SapFlatField constructor), then the value is set according to the acquisition device video type when calling the Create method.

If the current flat-field object is not associated with a SapAcquisition object, then the object will be used only for offline operation (no acquisition device), so that only software correction will be available. In this case, you should call SetVideoType before the Create method.

SapFlatField | SapFlatField Class Members | Basic Class Hierarchy Chart

SapFlatField::GetDeviationMaxBlack, SapFlatField::SetDeviationMaxBlack

int **GetDeviationMaxBlack**();

virtual BOOL **SetDeviationMaxBlack**(int *deviationMax*);

Remarks

Gets/sets the maximum deviation of the calculated coefficients from the average value towards the black pixel value so a pixel is not considered as being defective

The initial value for this attribute is 0. It is then set to 25% of the highest possible pixel value when calling the Create method. This pixel value is calculated either from the acquisition device pixel depth, or from the input buffer pixel depth, depending on which version of the SapFlatField constructor was used.

The maximum deviation value is used when calculating flat-field correction gain coefficients with the ComputeGain method.

SapFlatField | SapFlatField Class Members | Basic Class Hierarchy Chart

SapFlatField::GetDeviationMaxWhite, SapFlatField::SetDeviationMaxWhite

int **GetDeviationMaxWhite**();

virtual BOOL **SetDeviationMaxWhite**(int *deviationMax*);

Remarks

Gets/sets the maximum deviation of the calculated coefficients from the average value towards the white pixel value so a pixel is not considered as being defective

The initial value for this attribute is 0. It is then set to 25% of the highest possible pixel value when calling the Create method. This pixel value is calculated either from the acquisition device pixel depth, or from the input buffer pixel depth, depending on which version of the SapFlatField constructor was used.

The maximum deviation value is used when calculating flat-field correction gain coefficients with the ComputeGain method.

SapFlatField | SapFlatField Class Members | Basic Class Hierarchy Chart

SapFlatField::GetGainDivisor, SapFlatField::SetGainDivisor

int **GetGainDivisor**();

virtual BOOL **SetGainDivisor**(int *gainDivisor*);

Remarks

Gets/sets the factor by which the gain coefficients have to be divided for getting a unitary scale factor.

The initial value for this attribute is 128. It is then set to the acquisition device gain divisor value when calling the Create method.

The SetGainDivisor method can only be used when operating without hardware support.

SapFlatField | SapFlatField Class Members | Basic Class Hierarchy Chart

SapFlatField::SetOffsetMinMax

virtual BOOL **SetOffsetMinMax**(int *offsetMin*, int *offsetMax*);

Remarks

Sets the minimum and maximum resulting values when computing offset values using the ComputeOffset method.

This is useful when computing the offset values for an acquisition device that has known limitations on these values.

The initial value for these attributes are 0 and 255.

SapFlatField | SapFlatField Class Members | Basic Class Hierarchy Chart

SapFlatField::SetGainMinMax

virtual BOOL **SetGainMinMax**(int *gainMin*, int *gainMax*);

Remarks

Sets the minimum and maximum resulting values when computing gain values using the ComputeGain method.

This is useful when computing the gain values for an acquisition device that has known limitations on these values.

The initial value for these attributes are 0 and 255.

SapFlatField | SapFlatField Class Members | Basic Class Hierarchy Chart

SapFlatField::GetNumLinesAverage, SapFlatField::SetNumLinesAverage

int **GetNumLinesAverage**();

virtual BOOL **SetNumLinesAverage**(int *numLinesAverage*);

Remarks

Gets/sets the number of lines to be averaged in the image used for doing the calibration before computing the flat-field correction gain and offset coefficients for linescan video source. The initial value for this attribute is 128.

SapFlatField | SapFlatField Class Members | Basic Class Hierarchy Chart

SapFlatField::GetVerticalOffset, SapFlatField::SetVerticalOffset

int **GetVerticalOffset**();

virtual BOOL **SetVerticalOffset**(int *verticalOffset*);

Parameters

verticalOffset Vertical offset in lines

Remarks

Gets/sets the vertical line scan averaging offset in a full frame.

The initial value for this attribute is 0. This means that, for line scan acquisition, correction is performed on all lines. Specify a nonzero value if you need to skip a fixed number of lines at the beginning of each frame.

SapFlatField | SapFlatField Class Members | Basic Class Hierarchy Chart

SapFlatField::GetIndex, SapFlatField::SetIndex

int **GetIndex**();

virtual BOOL **SetIndex**(int *index*);

Parameters

index Gain and offset coefficients buffer pair index

Remarks

Gets/sets the current buffer index for gain and offset coefficients. See the SetNumFlatField method for details.

The initial value for this attribute is 0. Calling the SetNumFlatField method also resets it to 0.

SapFlatField | SapFlatField Class Members | Basic Class Hierarchy Chart

SapFlatField::Next

virtual void **Next**();

Remarks

Increments the index of the current buffer pair for gain and offset coefficients.

Use this method for software correction only. Call it in application code right before calling the Execute method. The index wraps around to 0 when it reaches its maximum value (specified using the constructor or the SetNumFlatField method).

For hardware correction, management of the current gain/offset buffer pair is automatically managed by the transfer device. See the SapXferPair::GetFlatFieldCycleMode, SapXferPair::SetFlatFieldCycleMode method for more details.

SapFlatField | SapFlatField Class Members | Basic Class Hierarchy Chart

SapFlatField::Load

virtual BOOL **Load**(const char **fileName*);

virtual BOOL **Load**(SapBuffer **pBufferGain*, SapBuffer **pBufferOffset*);

Parameters

fileName	Name of the image file with the gain and offset parameters
pBufferGain	Pointer to buffer object containing the gain values
pBufferOffset	Pointer to buffer object containing the offset values

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Loads flat-field correction gain and offset coefficients buffers from disk files or from existing buffer objects.

The specified file is in TIFF format, and contains the data for both buffers.

SapFlatField | SapFlatField Class Members | Basic Class Hierarchy Chart

SapFlatField::Save

virtual BOOL **Save**(const char **fileName*);

Parameters

fileName	Name of the image file with the gain and offset parameters

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Saves flat-field correction gain and offset coefficients buffers to disk files. The specified file is always written in TIFF format, no matter which file extension you specify.

SapFlatField | SapFlatField Class Members | Basic Class Hierarchy Chart

SapFlatField::Clear

virtual BOOL **Clear**();

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Clears the flat-field correction gain and offset coefficients buffers. The gain coefficients are initialized for getting a unitary scale factor while the offset coefficients are initialized to 0.

SapFlatField | SapFlatField Class Members | Basic Class Hierarchy Chart

SapFlatField::ComputeOffset

virtual BOOL **ComputeOffset**(SapBuffer **pBuffer*);

Parameters

pBuffer	Pointer to buffer object containing a calibration image

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Calculates the flat-field correction offset coefficients from a calibration image. This method has to be called before calling the ComputeGain method.

SapFlatField | SapFlatField Class Members | Basic Class Hierarchy Chart

SapFlatField::ComputeGain

virtual BOOL **ComputeGain**(SapBuffer **pBuffer*, SapFlatFieldDefects **pDefects*,
BOOL *bUseImageMaxValue* = TRUE);

Parameters

pBuffer	Pointer to a buffer object containing a calibration image
pDefects	Pointer to a SapFlatFieldDefects object
bUseImageMaxValue	Determines how to calculate the range of the calibrated output images

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Calculates the flat-field correction gain coefficients from a calibration image.

If *bUseImageMaxValue* is TRUE, then ComputeGain uses the highest actual pixel value of the input buffer as the maximum output value. Otherwise, it uses the highest possible pixel value, according to the pixel depth (see the SapBuffer::GetPixelDepth, SapBuffer::SetPixelDepth method).

When the method call returns, the SapFlatFieldDefects object pointed to by *pDefects* contains statistics about the defects found in the gain image. It has the following attributes:

int GetNumDefects()	Returns number of defective pixels
int GetNumClusters()	Returns number of defective pixels that are adjacent
float GetDefectRatio()	Returns ratio between defective pixels and good pixels in percent

SapFlatField | SapFlatField Class Members | Basic Class Hierarchy Chart

SapFlatField::Enable

virtual BOOL **Enable**(BOOL *enable* = TRUE, BOOL *useHardware* = TRUE);

Parameters

enable	TRUE to enable flat-field correction, FALSE to disable it
useHardware	TRUE to use hardware correction, FALSE to use the software implementation

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Enables/disables flat-field correction. If you set *useHardware* to TRUE and hardware correction is not available, then this method returns FALSE. If you set *useHardware* to FALSE, then you must call the Execute method to perform the actual correction.

SapFlatField | SapFlatField Class Members | Basic Class Hierarchy Chart

SapFlatField::EnablePixelReplacement

virtual BOOL **EnablePixelReplacement**(BOOL *enable* = TRUE);

Parameters

enable TRUE to enable pixel replacement, FALSE to disable it

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Enables/disables replacement of defective pixels.

Pixel replacement is used when calling the Execute method to perform the software implementation of flat-field correction. If TRUE, then defective pixel values are replaced by the value of a neighboring pixel. This is usually the one to the left of the current pixel, except for the first column, where the value of the pixel to the right is used instead.

SapFlatField | SapFlatField Class Members | Basic Class Hierarchy Chart

SapFlatField::Execute

virtual BOOL **Execute**(SapBuffer **pBuffer*);

virtual BOOL **Execute**(SapBuffer **pBuffer*, int *bufIndex*);

Parameters

pBuffer Pointer to a buffer object for performing flat-field correction

bufIndex Buffer resource index

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Performs the software implementation of flat-field correction. If no buffer index is specified, the current index is assumed.

SapFlatField | SapFlatField Class Members | Basic Class Hierarchy Chart

SapFlatField::GetAverage

virtual BOOL **GetAverage**(SapBuffer **pBuffer*, SapFlatFieldStats **pStats*);

Parameters

pBuffer	Pointer to buffer object
pStats	Pointer to a SapFlatFieldStats object

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets average pixel value and standard deviation for a buffer. See the GetStats method for details about the SapFlatFieldStats class.

SapFlatField | SapFlatField Class Members | Basic Class Hierarchy Chart

SapFlatField::GetStats

virtual BOOL **GetStats**(SapBuffer **pBuffer*, SapFlatFieldStats **pStats*);

Parameters

pBuffer	Pointer to a buffer object
pStats	Pointer to a SapFlatFieldStats object

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Calculates statistics about the image that are used internally to compute the flat-field correction gain and offset coefficients.

When the method call returns, the SapFlatFieldStats object pointed to by *pStats* contains statistics about the image. It has the following attributes:

int GetNumComponents()	Returns the number of color components for which statistics were computed. For a monochrome image, it is 1. For a Bayer image, it is 4, corresponding to the components of the Bayer scheme of the camera (see the SapBayer::GetAlign, SapBayer::SetAlign method).
int GetAverage()	Returns buffer average
int GetStdDeviation()	Returns buffer standard deviation
int GetPeakPosition()	Returns the peak value position in the histogram used to determine the gain coefficients
int GetLow()	Returns the lower bound of the histogram. Pixels below the lower

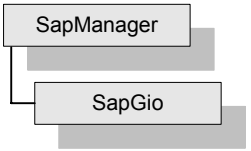
	bound will be assigned a gain of 2.
int GetHigh()	Returns the higher bound of the histogram. Pixels above the higher bound will be assigned a gain of 1.
int GetNumPixels()	Returns the number of pixels in the histogram between the lower and the higher bounds
float GetPixelRatio()	Returns the ratio between the number of pixels inside the lower and the higher bound of the histogram and the number of pixels in the buffer in percent

All methods except GetNumComponents accept an optional iComponent argument that specifies the component index for which statistics are retrieved. If not specified, the value of this argument is 0, corresponding to the first component.

Note that only the GetNumComponents, GetAverage, and GetStdDeviation methods are relevant when the SapFlatFieldStats object is used in a call to the GetAverage method.

SapFlatField | SapFlatField Class Members | Basic Class Hierarchy Chart

SapGio



The purpose of the SapGio Class is to control a block of general inputs and outputs, that is, a group of I/Os that may be read and/or written all at once. For a TTL level type I/Os, its state is considered ON or active if the measured voltage on the I/O is 5V (typical).

This class may be used together with SapCounter to associate event counting with the state of specific I/O pins.

Note that acquisition devices do not all support general I/Os.

#include <SapClassBasic.h>

SapGio Class Members | Base Class: SapManager | Basic Class Hierarchy Chart

SapGio Class Members

Construction	
SapGio	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Attributes	
GetLocation	Returns the location where the I/O resource is located
SetCallbackInfo	Sets the application callback method for I/O events and the associated context
GetCallback	Gets the current application callback method for I/O events
GetContext	Gets the application context associated with I/O events
GetNumPins	Gets the number of pins present on the I/O resource
GetAvailPinConfig	Gets the set of possible configurations for a specific I/O pin or all pins
GetPinConfig	Gets/sets the current configuration for a specific I/O pin or all pins

SetPinConfig	
GetPinState SetPinState	Gets/sets the low/high state of a specific I/O pin or all pins
GetHandle	Gets the low-level Sapera handle of the I/O resource
Operations	
EnableCallback	Allows an application callback function to be called at specific I/O events
DisableCallback	Disables calls to the application callback function
AutoTrigger	Automatically changes the state of an I/O pin for a specified duration
IsCapabilityValid	Checks for the availability of a low-level Sapera C library capability
IsParameterValid	Checks for the availability of a low-level Sapera C library parameter
GetCapability	Gets the value of a low-level Sapera C library capability
GetParameter SetParameter	Gets/sets the value of a low-level Sapera C library parameter

SapGio | Base Class: SapManager | Basic Class Hierarchy Chart

Member Functions

The following functions are members of the SapGio Class.

SapGio::SapGio

```
SapGio(
  SapLocation loc = SapLocation::ServerSystem,
  SapGioCallback pCallback = NULL,
  void *pContext = NULL
);
```

Parameters

loc	SapLocation object specifying the server where the I/O resource is located and the index of the resource on this server
pCallback	Application callback function to be called each time an I/O event happens. The callback function must be declared as: void MyCallback(SapGioCallbackInfo *pInfo);
pContext	Optional pointer to an application context to be passed to the callback function. If *pCallback* is NULL, this parameter is ignored.

Remarks

The SapGio constructor does not actually create the low-level Sapera resources. To do this, you must call the Create method.

Specifying a callback function in the constructor does not automatically activate it after the call to the Create method. You must subsequently call the EnableCallback method in order to be notified of I/O events.

SapGio | SapGio Class Members | Basic Class Hierarchy Chart

SapGio::Create

virtual BOOL **Create**();

Return Value

Returns TRUE if the object was successfully created, FALSE otherwise

Remarks

Creates all the low-level Sapera resources needed by the I/O object

SapGio | SapGio Class Members | Basic Class Hierarchy Chart

SapGio::Destroy

virtual BOOL **Destroy**();

Return Value

Returns TRUE if the object was successfully destroyed, FALSE otherwise

Remarks

Destroys all the low-level Sapera resources needed by the I/O object

SapGio | SapGio Class Members | Basic Class Hierarchy Chart

SapGio::GetLocation

SapLocation **GetLocation**();

Remarks

Returns the location where the I/O resource is located, as specified in the SapGio constructor.

SapGio | SapGio Class Members | Basic Class Hierarchy Chart

SapGio::SetCallbackInfo

virtual BOOL **SetCallbackInfo**(SapGioCallback *pCallback*, void ***pContext** = NULL);

Remarks

Sets the application callback method for I/O events and the associated context. See the SapGio constructor for more details.

You can only call SetCallbackInfo before the EnableCallback method.

SapGio | SapGio Class Members | Basic Class Hierarchy Chart

SapGio::GetCallback

SapGioCallback **GetCallback**();

Remarks

Gets the current application callback method for I/O events. The initial value for this attribute is NULL, unless you specify another value in the constructor.

See the SapGio constructor for more details.

SapGio | SapGio Class Members | Basic Class Hierarchy Chart

SapGio::GetContext

void ***GetContext**();

Remarks

Gets the application context associated with I/O events. The initial value for this attribute is NULL, unless you specify another value in the constructor.

See the SapGio constructor for more details.

SapGio | SapGio Class Members | Basic Class Hierarchy Chart

SapGio::GetNumPins

int **GetNumPins**();

Remarks

Gets the number of pins present on the I/O resource.

The initial value for this attribute is 0. It is then set to the I/O device pin count value when calling the Create method.

SapGio | SapGio Class Members | Basic Class Hierarchy Chart

SapGio::GetAvailPinConfig

virtual BOOL **GetAvailPinConfig**(int *pinNumber*, SapGio::PinConfig **pAvailPinConfig*);

virtual BOOL **GetAvailPinConfig**(SapGio::PinConfig **pAvailPinConfig*);

Parameters

pinNumber	Pin number on the current I/O resource	
pAvailPinConfig	Pointer to available pin configurations, including one or more of the following (combined using bitwise OR)	
	SapGio::PinInput	I/O pin may be configured as an input
	SapGio::PinOutput	I/O pin may be configured as an output
	SapGio::PinTristate	I/O pin may be tri-stated
	If no *pinNumber* is specified, then this argument must point to a memory area large enough to hold the values for all pins, as found by calling the GetNumPins method.	

Remarks

Gets the set of possible configurations for a specific I/O pin or all pins. The first form of this method takes a single pin number, and returns a single value through the *pAvailPinConfig* argument. The second form returns the configuration for all pins in the *pAvailPinConfig* array.

You can only call GetAvailPinConfig after the Create method

SapGio | SapGio Class Members | Basic Class Hierarchy Chart

SapGio::GetPinConfig, SapGio::SetPinConfig

virtual BOOL **GetPinConfig**(int *pinNumber*, SapGio::PinConfig **pPinConfig*);

virtual BOOL **GetPinConfig**(SapGio::PinConfig **pPinConfig*);

virtual BOOL **SetPinConfig**(int *pinNumber*, SapGio::PinConfig *pinConfig*);

virtual BOOL **SetPinConfig**(SapGio::PinConfig **pPinConfig*);

Parameters

pinNumber	Pin number on the current I/O resource
pPinConfig	Pointer to pin configuration. See the SapGio::GetAvailPinConfig method for possible values. If no *pinNumber* is specified, then this argument must point to a memory area large enough to hold the values for all pins, as found by calling the GetNumPins method.
pinConfig	New pin configuration. See the SapGio::GetAvailPinConfig method for possible values.

Remarks

Gets/sets the current configuration for a specific I/O pin or all pins.

The first form of GetPinConfig takes a single pin number and returns a single value through the *pPinConfig* argument. The second form returns the configuration for all pins in the *pPinConfig* array.

The first form of SetPinConfig may be used using a single pin number and a corresponding pin configuration. You may also set *pinNumber* to the special constant SapGio::AllPins to apply the specified *pinConfig* to all I/O pins. The second form of SetPinConfig allows all pins to be set to a different value through the *pPinConfig* array argument.

You can only call GetPinConfig and SetPinConfig after the Create method.

SapGio | SapGio Class Members | Basic Class Hierarchy Chart

SapGio::GetPinState, SapGio::SetPinState

virtual BOOL **GetPinState**(int *pinNumber*, SapGio::PinState **pPinState*);

virtual BOOL **GetPinState**(SapGio::PinState **pPinState*);

virtual BOOL **SetPinState** (int *pinNumber*, SapGio::PinState *pinState*);

virtual BOOL **SetPinState** (int *pinMask*, SapGio::PinState **pPinState*);

Parameters

pinNumber	Pin number on the current I/O resource
pPinState	Pointer to pin state, can be one of the following values: SapGio::PinLow The I/O pin is low SapGio::PinHigh The I/O pin is high If no *pinNumber* is specified in GetPinState, then this argument must point to a memory area large enough to hold the values for all pins, as found by calling the GetNumPins method.
pinState	New pin state. See above for possible values.
pinMask	Bit mask specifying which I/O pins will be affected. The least significant bit corresponds to pin 0, the next bit corresponds to pin 1, and so on. Each bit set to 1 enables the corresponding pin.

Remarks

Gets/sets the current state of a specific I/O pin or all pins.

The first form of GetPinState takes a single pin number and returns a single value through the *pPinState* argument. The second form returns the configuration for all pins in the *pPinState* array.

The first form of SetPinState may be used using a single pin number and a corresponding pin state. The second form takes a bit mask of affected I/O pins, and an array *pPinState* to specify the state of each pin. Only entries in *pPinState* corresponding to bits set to 1 in the *pinMask* argument are used.

You can only call GetPinState and SetPinState after the Create method.

SapGio | SapGio Class Members | Basic Class Hierarchy Chart

SapGio::GetHandle

virtual CORHANDLE **GetHandle**();

Remarks

Gets the low-level Sapera handle of the I/O resource, which you may then use from the low-level Sapera functionality. The handle is only valid after you call the Create method.

See the *Sapera LT Basic Modules Reference Manual* for details on low-level Sapera functionality.

SapGio | SapGio Class Members | Basic Class Hierarchy Chart

SapGio::EnableCallback

virtual BOOL **EnableCallback**(int *pinNumber*, SapGio::EventType *eventType*);

virtual BOOL **EnableCallback**(int *pinMask*, SapGio::EventType **pEventType*);

virtual BOOL **EnableCallback**(SapGio::EventType *eventType*);

Parameters

pinNumber	Pin number on the current I/O resource	
eventType	Type of I/O event that initiates calls to the application callback function, can be one of the following values:	
	SapGio::EventRisingEdge	Rising edge of I/O pin state transition (low to high)
	SapGio::EventFallingEdge	Falling edge of I/O pin state transition (high to low)
pinMask	Bit field specifying which I/O pins will be affected. The least significant bit corresponds to pin 0, the next bit corresponds to pin 1, and so on. Each bit set to 1 enables the corresponding pin.	
pEventType	Pointer to event types array. This argument must point to a memory area large enough to hold the values for all pins, as found by calling the GetNumPins method.	

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Allows an application callback function to be called at specific I/O events. See the SapGio constructor for details about the application callback function.

The first form of EnableCallback may be used using a single pin number and a corresponding event type.

The second form takes a bit mask of affected I/O pins, and an array *pEventType* to specify the event associated with each pin. Entries in *pEventType* corresponding to bits set to 1 in the *pinMask* argument enable callbacks for the corresponding pins. Bits set to 0 in *pinMask* disable callbacks for the corresponding pins.

The third form enables callbacks for all pins using the same event type. The drawback of using this form is that it will not be possible to uniquely identify the pin causing the I/O event when the callback function is called.

SapGio | SapGio Class Members | Basic Class Hierarchy Chart

SapGio::DisableCallback

virtual BOOL **DisableCallback**(int *pinNumber*);

virtual BOOL **DisableCallback**();

Parameters

pinNumber Pin number on the current I/O resource.

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Disables calls to the application callback function.

The first form of Disable callback only affects the specified I/O pin. The second form affects all pins.

See the SapGio constructor and the EnableCallback method for more details.

SapGio | SapGio Class Members | Basic Class Hierarchy Chart

SapGio::AutoTrigger

virtual BOOL **AutoTrigger**(SapCounter **pCounter*, int *startCount*, int *stopCount*, int *pinMask*, int *pinState*);

Parameters

pCounter	Counter object that causes I/O state transitions when reaching *startCount* and *stopCount*
startCount	Count at which the I/O pins identified by *pinMask* will change state
stopCount	Count at which the I/O pins identified by *pinMask* will go back to their original state
pinMask	Bit field specifying which I/O pins will be affected. The least significant bit corresponds to pin 0, the next bit corresponds to pin 1, and so on. Each bit set to 1 enables the corresponding pin.
pinState	Bit field representing the state of I/O pins identified by *pinMask* when the counter resource reaches *startCount*. The least significant bit corresponds to pin 0, the next bit corresponds to pin 1, and so on. Bits that are set to 1 represent high, while 0 represents low.

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Automatically triggers the state of one or more I/O pins at a specific time.

The I/O pins identified by *pinMask* are initially set to the opposite of the values represented by *pinState*. When the counter device reaches *startCount*, their state changes to the values represented by *pinState*. When the counter device reaches *stopCount*, their state goes back to the original values.

SapGio | SapGio Class Members | Basic Class Hierarchy Chart

SapGio::IsCapabilityValid

virtual BOOL **IsCapabilityValid**(int *cap*);

Parameters

cap Low-level Sapera C library capability to be checked

Return Value

Returns TRUE if the capability is supported, FALSE otherwise

Remarks

Checks for the availability of a low-level Sapera C library capability for the general I/O module. Call this method before GetCapability to avoid invalid or not available capability errors.

IsCapabilityValid is rarely needed. The SapGio class already uses important capabilities internally for self-configuration and validation.

See the *Sapera LT Basic Modules Reference Manual* for a description of all capabilities and their possible values.

SapGio | SapGio Class Members | Basic Class Hierarchy Chart

SapGio::IsParameterValid

virtual BOOL **IsParameterValid**(int *param*);

Parameters

param Low-level Sapera C library parameter to be checked

Return Value

Returns TRUE if the parameter is supported, FALSE otherwise

Remarks

Checks for the availability of a low-level Sapera C library parameter for the general I/O module. Call this method before GetParameter to avoid invalid or not available parameter errors.

IsParameterValid is rarely needed. The SapGio class already uses important parameters internally for self-configuration and validation.

See the *Sapera LT Basic Modules Reference Manual* for a description of all parameters and their possible values.

SapGio | SapGio Class Members | Basic Class Hierarchy Chart

SapGio::GetCapability

virtual BOOL **GetCapability**(int *cap*, void **pValue*);

Parameters

param	Low-level Sapera C library capability to read
pValue	Pointer to capability value to read back

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

This method allows direct read access to low-level Sapera C library capabilities for the I/O module. It needs a pointer to a memory area large enough to receive the capability value, which is usually a 32-bit integer.

You will rarely need to use GetCapability. The SapGio Class already uses important capabilities internally for self-configuration and validation.

See the *Sapera LT Basic Modules Reference Manual* for a description of all capabilities and their possible values.

SapGio | SapGio Class Members | Basic Class Hierarchy Chart

SapGio::GetParameter, SapGio::SetParameter

virtual BOOL **GetParameter**(int *param*, void **pValue*);

virtual BOOL **SetParameter**(int *param*, int *value*);

virtual BOOL **SetParameter**(int *param*, void **pValue*);

Parameters

param	Low-level Sapera C library parameter to read or write
pValue	Pointer to parameter value to read back or to write
value	New parameter value to write

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

These methods allow direct read/write access to low-level Sapera C library parameters for the I/O module. The GetParameter method needs a pointer to a memory area large enough to receive the parameter value, which is usually a 32-bit integer. The first form of SetParameter accepts a 32-bit value for the new value. The second form takes a pointer to the new value, and is required when the parameter uses more than 32-bits of storage.

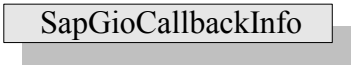
Note that you will rarely need to use these methods. You should first make certain that what you need is not

already supported through the SapGio Class. Also, directly setting parameter values may interfere with the correct operation of the class.

See the *Sapera LT Basic Modules Reference Manual* for a description of all parameters and their possible values.

SapGio | SapGio Class Members | Basic Class Hierarchy Chart

SapGioCallbackInfo



The SapGioCallbackInfo Class acts as a container for storing all arguments to the callback function for the SapGio Class.

#include <SapClassBasic.h>

SapGioCallbackInfo Class Members | Basic Class Hierarchy Chart

SapGioCallbackInfo Class Members

Construction	
SapGioCallbackInfo	Class constructor
Attributes	
GetGio	Gets the SapGio object associated with I/O events
GetEventType	Gets the I/O events that triggered the call to the application callback
GetEventCount	Gets the current count of I/O events
GetContext	Gets the application context associated with I/O events
GetPinNumber	Get the I/O pin number that generated an I/O event

SapGioCallbackInfo | Basic Class Hierarchy Chart

Member Functions

The following functions are members of the SapGioCallbackInfo Class.

SapGioCallbackInfo::SapGioCallbackInfo

SapGioCallbackInfo(
  SapGio **pGio*,
  void **pContext*,
  SapGio::EventType *eventType*,
  int *eventCount*
...int *pinNumber*
);

Parameters

pGio	SapGio object that calls the callback function
pContext	Pointer to the application context
eventType	Combination of I/O events. See SapGio::EnableCallback for a list a possible values
eventCount	Current I/O event count
pinNumber	Current I/O pin number

Remarks

SapGio objects create an instance of this class before each call to the I/O callback method, in order to combine all function arguments into one container.

SapGio uses this class for reporting of I/O events. The *pContext* parameter takes the value specified in the SapGio Class constructor, *eventType* identifies the combination of events that triggered the call to the callback function, *eventCount* increments by one at each call (starting at 1), and *pinNumber* identifies the I/O pin that had a state change.

SapGioCallbackInfo | SapGioCallbackInfo Class Members | Basic Class Hierarchy Chart

SapGioCallbackInfo::GetGio

SapGio ***GetGio**();

Remarks

Gets the SapGio object associated with I/O events. See the SapGio constructor for more details.

SapGioCallbackInfo | SapGioCallbackInfo Class Members | Basic Class Hierarchy Chart

SapGioCallbackInfo::GetEventType

SapGio::EventType **GetEventType**();

Remarks

Gets the combination of I/O events that triggered the call to the application callback. See the SapGio constructor for the list of possible values.

SapGioCallbackInfo | SapGioCallbackInfo Class Members | Basic Class Hierarchy Chart

SapGioCallbackInfo::GetEventCount

int **GetEventCount**();

Remarks

Gets the current count of I/O events. The initial value is 1, and increments after every call to the I/O callback function.

SapGioCallbackInfo | SapGioCallbackInfo Class Members | Basic Class Hierarchy Chart

SapGioCallbackInfo::GetContext

void ***GetContext**();

Remarks

Gets the application context associated with I/O events. See the SapGio constructor for more details.

SapGioCallbackInfo | SapGioCallbackInfo Class Members | Basic Class Hierarchy Chart

SapGioCallbackInfo::GetPinNumber

int **GetPinNumber**();

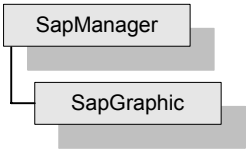
Remarks

Get the I/O pin number that generated an I/O event.

If this number is equal to the special constant SapGio::AllPins, the pin then cannot be uniquely identified. In this case, use the SapGio::GetState method to get the required pin information.

SapGioCallbackInfo | SapGioCallbackInfo Class Members | Basic Class Hierarchy Chart

SapGraphic



The SapGraphic Class implements the drawing of graphic primitives and text strings. It supports these operations either destructively on image data itself (using the low-level Sapera graphic module), or in non-destructive overlay over displayed images (using Windows GDI functions).

If you need more advanced graphic capabilities in non-destructive overlay, you will have to use GDI directly. You will also have to call the SapView::GetDC and SapView::ReleaseDC methods to first obtain a valid Windows display context, and then release it when you are done.

#include <SapClassBasic.h>

SapGraphic Class Members | Base Class: SapManager | Basic Class Hierarchy Chart

SapGraphic Class Members

Construction	
SapGraphic	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Attributes	
GetLocation	Returns the location where the graphic resource is located
GetDrawMode SetDrawMode	Gets/sets the current foreground drawing mode
GetTransparency SetTransparency	Gets/sets the current transparency mode relative to the background
GetColor SetColor	Gets/sets the current foreground drawing color

GetBackColor SetBackColor	Gets/sets the current background drawing color
GetTextAlign SetTextAlign	Gets/sets the current horizontal text alignment mode
GetHandle	Gets the low-level Sapera handle of the graphic resource
Operations	
Clear	Clears the drawing area
Circle	Draws a circle
Ellipse	Draws an ellipse
Dot	Draws a single dot
Line	Draws a line
Rectangle	Draws a rectangle
Text	Draws a text string
SetBatchMode	Allows delayed screen update of drawing commands in non-destructive overlay
Flush	Updates non-destructive overlay with accumulated drawing commands
GetCapability	Gets the value of a low-level Sapera C library capability
GetParameter SetParameter	Gets/sets the value of a low-level Sapera C library parameter

SapGraphic | Base Class: SapManager | Basic Class Hierarchy Chart

Member Functions

The following functions are members of the SapGraphic Class

SapGraphic::SapGraphic

SapGraphic(
  SapLocation *loc* = SapLocation::ServerSystem,
);

Parameters

loc SapLocation object specifying the server where the graphic resource is located and the index of the resource on this server

Remarks

The SapGraphic constructor does not actually create the low-level Sapera resources. To do this, you must call the Create method.

SapGraphic | SapGraphic Class Members | Basic Class Hierarchy Chart

SapGraphic::Create

virtual BOOL **Create**();

Return Value

Returns TRUE if the object was successfully created, FALSE otherwise

Remarks

Creates all the low-level Sapera resources needed by the graphic object

SapGraphic | SapGraphic Class Members | Basic Class Hierarchy Chart

SapGraphic::Destroy

virtual BOOL **Destroy**();

Return Value

Returns TRUE if the object was successfully destroyed, FALSE otherwise

Remarks

Destroys all the low-level Sapera resources needed by the graphic object

SapGraphic | SapGraphic Class Members | Basic Class Hierarchy Chart

SapGraphic::GetLocation

SapLocation **GetLocation**();

Remarks

Returns the location where the graphic resource is located, as specified in the SapGraphic constructor.

SapGraphic | SapGraphic Class Members | Basic Class Hierarchy Chart

SapGraphic::GetDrawMode, SapGraphic::SetDrawMode

SapGraphic::DrawMode **GetDrawMode**();

virtual BOOL **SetDrawMode**(SapGraphic::DrawMode *drawMode*);

Parameters

drawMode New drawing mode that specifies how the foreground color and the existing color on the drawing surface are combined together. The following values are allowed:

SapGraphic::ModeReplace	Use the foreground color only
SapGraphic::ModeAnd	Use bitwise AND between the two colors
SapGraphic::ModeOr	Use bitwise OR between the two colors
SapGraphic::ModeXor	Use bitwise XOR between the two colors

Remarks

Gets/sets the current foreground drawing mode that applies all shape drawing methods, but not text.

The initial value for this attribute is ModeReplace, unless you construct this object using an existing SapGraphic object.

You can only call SetDrawMode before the Create method.

SapGraphic | SapGraphic Class Members | Basic Class Hierarchy Chart

SapGraphic::GetTransparency, SapGraphic::SetTransparency

BOOL **GetTransparency**();

virtual BOOL **SetTransparency**(BOOL *isTransparent*);

Remarks

Gets/sets the current transparency mode relative to the background. When transparency is active, the existing background content is unaffected when drawing text strings. When transparency is off, the current background drawing color is used instead.

The initial value for this attribute is FALSE.

Transparency does not apply to the drawing of graphic shapes.

You can only call SetTransparency before the Create method.

SapGraphic | SapGraphic Class Members | Basic Class Hierarchy Chart

SapGraphic::GetColor, SapGraphic::SetColor

SapData **GetColor**();

virtual BOOL **SetColor**(SapData *color*);

Remarks

Gets/sets the current foreground drawing color. For a monochrome drawing surface, this involves a SapDataMono object. For a color surface, this involves a SapDataRGB object.

The initial value for this attribute is white, unless you construct this object using an existing SapGraphic object.

You can only call SetColor before the Create method.

SapGraphic | SapGraphic Class Members | Basic Class Hierarchy Chart

SapGraphic::GetBackColor, SapGraphic::SetBackColor

SapData **GetBackColor**();

virtual BOOL **SetBackColor**(SapData *backColor*);

Remarks

Gets/sets the current background drawing color. For a monochrome drawing surface, this involves a SapDataMono object. For a color surface, this involves a SapDataRGB object.

The initial value for this attribute is black, unless you construct this object using an existing SapGraphic object.

The background color applies to text only, not to the drawing of graphic shapes.

You can only call SetBackColor before the Create method.

SapGraphic | SapGraphic Class Members | Basic Class Hierarchy Chart

SapGraphic::GetTextAlign, SapGraphic::SetTextAlign

SapGraphic::TextAlign **GetTextAlign**();

virtual BOOL **SetTextAlign**(SapGraphic::TextAlign *textAlign*);

Parameters

textAlign New text alignment mode that specifies where text strings are drawn relative to their starting (x, y) coordinates. The following values are allowed:

SapGraphic::TextLeft	Coordinates represent left side of text string
SapGraphic::TextCenter	Coordinates represent middle of text string
SapGraphic::TextRight	Coordinates represent right side of text string

Remarks

Gets/sets the current horizontal text alignment mode. This does not apply to graphic shapes.

The initial value for this attribute is TextLeft, unless you construct this object using an existing SapGraphic object.

You can only call SetDrawMode before the Create method.

SapGraphic | SapGraphic Class Members | Basic Class Hierarchy Chart

SapGraphic::GetHandle

virtual CORHANDLE **GetHandle**();

Remarks

Gets the low-level Sapera handle of the graphic resource that you may then use from the low-level Sapera functionality. The handle is only valid after you call the Create method.

See the *Sapera LT Basic Modules Reference Manual* for details on low-level Sapera functionality.

SapGraphic | SapGraphic Class Members | Basic Class Hierarchy Chart

SapGraphic::Clear

BOOL **Clear**(SapBuffer **pBuffer*);

BOOL **Clear**(SapView **pView*);

Parameters

pBuffer	Buffer object to use when drawing in image data. The current buffer index is assumed.
pView	View object to use when drawing in non-destructive image overlay

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Clears the drawing area using the current foreground color and drawing mode.

Drawing in non-destructive overlay is only possible if the SapBuffer object associated with *pView* has the SapBuffer::TypeOverlay type.

SapGraphic | SapGraphic Class Members | Basic Class Hierarchy Chart

SapGraphic::Circle

BOOL **Circle**(SapBuffer **pBuffer*, int *x*, int *y*, int *radius*, BOOL *fill* = FALSE);

BOOL **Circle**(SapView **pView*, int *x*, int *y*, int *radius*, BOOL *fill* = FALSE);

Parameters

pBuffer	Buffer object to use when drawing in image data. The current buffer index is assumed.
pView	View object to use when drawing in non-destructive image overlay
x	Horizontal coordinate of circle origin
y	Vertical coordinate of circle origin
radius	Radius of circle (in pixels)
fill	Specifies whether a filled shape should be drawn

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Draws a circle at (*x*, *y*) with the specified *radius*. The current foreground color and drawing mode are used.

If *fill* is TRUE, the whole area covered by the circle is filled. If FALSE, only the outline is drawn.

Drawing in non-destructive overlay is only possible if the SapBuffer object associated with *pView* has the SapBuffer::TypeOverlay type.

SapGraphic | SapGraphic Class Members | Basic Class Hierarchy Chart

SapGraphic::Ellipse

BOOL **Ellipse**(SapBuffer **pBuffer*, int *x*, int *y*, int *xRadius*, int *yRadius*, BOOL *fill* = FALSE);

BOOL **Ellipse**(SapView **pView*, int *x*, int *y*, int *xRadius*, int *yRadius*, BOOL *fill* = FALSE);

Parameters

pBuffer	Buffer object to use when drawing in image data. The current buffer index is assumed.
pView	View object to use when drawing in non-destructive image overlay
x	Horizontal coordinate of ellipse origin
y	Vertical coordinate of ellipse origin
xRadius	Horizontal radius of ellipse (in pixels)
yRadius	Vertical radius of ellipse (in lines)
fill	Specifies whether a filled shape should be drawn

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Draws an ellipse at (*x*, *y*) with the specified *xRadius* and *yRadius*. The current foreground color and drawing mode are used.

If *fill* is TRUE, the whole area covered by the ellipse is filled. If FALSE, only the outline is drawn.

Drawing in non-destructive overlay is only possible if the SapBuffer object associated with *pView* has the SapBuffer::TypeOverlay type.

SapGraphic | SapGraphic Class Members | Basic Class Hierarchy Chart

SapGraphic::Dot

BOOL **Dot**(SapBuffer **pBuffer*, int *x*, int *y*);

BOOL **Dot**(SapView **pView*, int *x*, int *y*);

Parameters

pBuffer	Buffer object to use when drawing in image data. The current buffer index is assumed.
pView	View object to use when drawing in non-destructive image overlay
x	Horizontal dot coordinate
y	Vertical dot coordinate

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Draws a single dot at (*x*, *y*). The current foreground color and drawing mode are used.

Drawing in non-destructive overlay is only possible if the SapBuffer object associated with *pView* has the SapBuffer::TypeOverlay type.

SapGraphic | SapGraphic Class Members | Basic Class Hierarchy Chart

SapGraphic::Line

BOOL **Line**(SapBuffer **pBuffer*, int *x1*, int *y1*, int *x2*, int *y2*);

BOOL **Line**(SapView **pView*, int *x1*, int *y1*, int *x2*, int *y2*);

Parameters

pBuffer	Buffer object to use when drawing in image data. The current buffer index is assumed.
pView	View object to use when drawing in non-destructive image overlay
x1	Starting horizontal coordinate
y1	Starting vertical coordinate
x2	Ending horizontal coordinate
y2	Ending vertical coordinate

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Draws a line from (*x1*, *y1*) to (*x2*, *y2*). The ending point at (*x2*, *y2*) is drawn. The current foreground color and drawing mode are used.

Drawing in non-destructive overlay is only possible if the SapBuffer object associated with *pView* has the SapBuffer::TypeOverlay type.

SapGraphic | SapGraphic Class Members | Basic Class Hierarchy Chart

SapGraphic::Rectangle

BOOL **Rectangle**(SapBuffer **pBuffer*, int *x1*, int *y1*, int *x2*, int *y2*, BOOL *fill* = FALSE);

BOOL **Rectangle** (SapView **pView*, int *x1*, int *y1*, int *x2*, int *y2*, BOOL *fill* = FALSE);

Parameters

pBuffer	Buffer object to use when drawing in image data. The current buffer index is assumed.
pView	View object to use when drawing in non-destructive image overlay
x1	Horizontal coordinate of top left corner
y1	Vertical coordinate of top left corner
x2	Horizontal coordinate of bottom right corner
y2	Vertical coordinate of bottom right corner
fill	Specifies whether a filled shape should be drawn

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Draws a rectangle with corners at (*x1*, *y1*) and (*x2*, *y2*). The corner at (*x2*, *y2*) is drawn. The current foreground color and drawing mode are used.

If *fill* is TRUE, the whole area covered by the rectangle is filled. If FALSE, only the outline is drawn.

Drawing in non-destructive overlay is only possible if the SapBuffer object associated with *pView* has the SapBuffer::TypeOverlay type.

SapGraphic | SapGraphic Class Members | Basic Class Hierarchy Chart

SapGraphic::Text

BOOL **Text**(SapBuffer **pBuffer*, int *x*, int *y*, const char **text*);

BOOL **Text**(SapView **pView*, int *x*, int *y*, const char **text*);

Parameters

pBuffer	Buffer object to used when drawing in image data. The current buffer index is assumed.
pView	View object to use when drawing in non-destructive image overlay
x	Horizontal text coordinate
y	Vertical text coordinate
text	Text string to draw

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Draws a text string at (*x*, *y*). The current foreground/background colors, transparency mode, and text alignment options are used.

Drawing in non-destructive overlay is only possible if the SapBuffer object associated with *pView* has the SapBuffer::TypeOverlay type.

SapGraphic | SapGraphic Class Members | Basic Class Hierarchy Chart

SapGraphic::SetBatchMode

BOOL **SetBatchMode**(BOOL *batchMode*, SapView **pView*);

Parameters

batchMode	TRUE to enable buffering of drawing commands, FALSE to disable it
pView	View object to use when drawing in non-destructive image overlay

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Allows delayed screen update of drawing commands in non-destructive overlay.

By default, drawing commands update the display as they are executed. When batch mode is active, these commands do not update the display immediately. Rather, they update an internally managed and invisible drawing area. It is then possible to update the display whenever needed by calling the Flush method.

This technique improves performance of screen updates, and may reduce flicker effects often associated with graphics.

Batch mode is only supported for the primary VGA board in the system. It is furthermore not supported when drawing in image data.

SapGraphic | SapGraphic Class Members | Basic Class Hierarchy Chart

SapGraphic::Flush

BOOL **Flush**(SapView **pView*, int *x1* = 0, int *y1* = 0, int *x2* = -1, int *y2* = -1);

Parameters

pView	View object to use when drawing in non-destructive image overlay
x1	Horizontal coordinate of top left corner of updated area
y1	Vertical coordinate of top left corner of updated area
x2	Horizontal coordinate of bottom right corner of updated area
y2	Vertical coordinate of bottom right corner of updated area

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Updates non-destructive overlay with accumulated drawing commands. The area from (*x1*, *y1*) to (*x2*, *y2*) of the internal drawing surface is copied to the display in one operation. The contents of the drawing surface remain unaffected and may be modified again so that you may call Flush later using the newest data.

When the update area is specified as (0, 0) to (-1, -1), the whole drawing area is copied to the display. This is the default behavior. Specifying a smaller area improves performance of screen updates.

Flush is only available in batch mode and is not supported when drawing in image data.

See the SapGraphic::SetBatchMode method for more details.

SapGraphic | SapGraphic Class Members | Basic Class Hierarchy Chart

SapGraphic::GetCapability

virtual BOOL **GetCapability**(int *cap*, void **pValue*);

Parameters

param	Low-level Sapera C library capability to read
pValue	Pointer to capability value to read back

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

This method allows direct read access to low-level Sapera C library capabilities for the graphic module. It needs a pointer to a memory area large enough to receive the capability value, which is usually a 32-bit integer.

You will rarely need to use GetCapability. The SapGraphic Class already uses important capabilities internally

for self-configuration and validation.

Calling GetCapability has no effect when using Windows GDI graphics.

See the *Sapera LT Basic Modules Reference Manual* for a description of all capabilities and their possible values.

SapGraphic | SapGraphic Class Members | Basic Class Hierarchy Chart

SapGraphic::GetParameter, SapGraphic::SetParameter

virtual BOOL **GetParameter**(int *param*, void **pValue*);

virtual BOOL **SetParameter**(int *param*, int *value*);

virtual BOOL **SetParameter**(int *param*, void **pValue*);

Parameters

param	Low-level Sapera C library parameter to read or write
pValue	Pointer to parameter value to read back or to write
value	New parameter value to write

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

These methods allow direct read/write access to low-level Sapera C library parameters for the graphic module. The GetParameter method needs a pointer to a memory area large enough to receive the parameter value, which is usually a 32-bit integer. The first form of SetParameter accepts a 32-bit value for the new value. The second form takes a pointer to the new value and is required when the parameter uses more than 32-bits of storage.

Note that you will rarely need to use these methods. You should first make certain that what you need is not already supported through the SapGraphic Class. Also, directly setting parameter values may interfere with the correct operation of the class.

Calling these methods has no effect when using Windows GDI graphics.

See the *Sapera LT Basic Modules Reference Manual* for a description of all parameters and their possible values.

SapGraphic | SapGraphic Class Members | Basic Class Hierarchy Chart

SapLocation

The SapLocation Class identifies a Sapera server/resource pair.

A Sapera server is an abstract representation of a physical device like a frame grabber, a processing board, or the host computer. In general, a DALSA board is a server. Resources are attached to these physical devices. For example, a frame grabber can have one or more acquisition resources, a display resource, and so on.

Sapera++ Class methods do not always need the server information from SapLocation. In these cases, the resource index is simply ignored.

#include <SapClassBasic.h>

SapLocation Class Members | Basic Class Hierarchy Chart

SapLocation Class Members

Construction	
SapLocation	Class constructor
Attributes	
GetServerIndex	Gets the server index
GetServerName	Gets the server name
GetResourceIndex	Gets the resource index
IsUnknown	Checks if neither the server index nor the server name is valid

SapLocation | Basic Class Hierarchy Chart

Member Functions

The following functions are members of the SapLocation Class.

SapLocation::SapLocation

```
SapLocation(
);
SapLocation(
  int serverIndex,
  int resourceIndex = 0
);
SapLocation(
  const char *serverName,
  int resourceIndex = 0
);
SapLocation(
  const SapLocation &loc
);
SapLocation(
  const SapLocation &loc,
  int resourceIndex
);
```

Parameters

serverIndex	Sapera server index. There is always one server associated with the host computer at SapLocation::ServerSystem (index 0).
serverName	Sapera server name. The 'System' server is associated with the host computer.
resourceIndex	Sapera resource index
loc	Existing SapLocation object from which server and resource information are to be extracted.

Remarks

Use the Sapera Configuration utility to find the names and indices of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names and resource indices for that product.

The constructor with only the *loc* argument allows you to reuse the same server and resource information. The constructor with both *loc* and *resourceIndex* allows use to reuse the same server information, while specifying a different resource index.

SapLocation | SapLocation Class Members | Basic Class Hierarchy Chart

SapLocation::GetServerIndex

int **GetServerIndex**();

Remarks

Returns the server index. If the returned value is equal to SapLocation::ServerUnknown, it does not necessarily mean that the object is invalid. In this case, use the GetServerName method instead.

SapLocation | SapLocation Class Members | Basic Class Hierarchy Chart

SapLocation::GetServerName

char ***GetServerName**();

Remarks

Returns the server name. If the returned value is an empty string, it does not necessarily mean that the object is invalid. In this case, use the GetServerIndex method instead.

SapLocation | SapLocation Class Members | Basic Class Hierarchy Chart

SapLocation::GetResourceIndex

int **GetResourceIndex**();

Remarks

Returns the resource index.

SapLocation | SapLocation Class Members | Basic Class Hierarchy Chart

SapLocation::IsUnknown

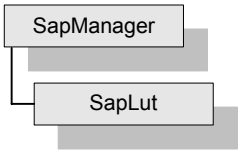
BOOL **IsUnknown**();

Remarks

Returns TRUE if neither the server index nor the server name is valid, FALSE otherwise. Although an unknown SapLocation is usually invalid, it may be useful in some particular cases.

SapLocation | SapLocation Class Members | Basic Class Hierarchy Chart

SapLut



The SapLut Class implements lookup table management. Although you may create and destroy SapLut objects explicitly in application code, you usually do not have to do this.

If you need to manipulate acquisition lookup tables, first call the SapAcquisition::GetLut method to get a valid SapLut object. You may then manipulate the LUT through the methods in this class, and reprogram it using the SapAcquisition::ApplyLut method.

If you need to manipulate display lookup tables, you may use the same technique, but using the SapView::GetLut and SapView::ApplyLut methods instead.

#include <SapClassBasic.h>

SapLut Class Members | Base Class: SapManager | Basic Class Hierarchy Chart

SapLut Class Members

Construction	
SapLut	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Attributes	
GetLocation	Returns the location where the LUT resource is located
GetNumEntries SetNumEntries	Gets/sets the number of LUT entries
GetFormat SetFormat	Gets/sets the LUT data format
GetElementSize	Gets the number of bytes required to store a single LUT element
GetNumPages	Gets the number of color planes in the LUT
IsSigned	Checks if the LUT contains signed or unsigned data

GetTotalSize	Gets the total number of bytes required to store LUT data
GetHandle	Gets the low-level Sapera handle of the LUT resource
Operations	
Read	Reads one or more elements from LUT storage to user-allocated memory
Write	Writes one or more elements from user-allocated memory to LUT storage
Arithmetic	Modifies all LUT entries using an arithmetic operation
BinaryPattern	Modifies some LUT entries based on a binary pattern
Boolean	Modifies all LUT entries using a Boolean operation
Gamma	Modifies all LUT entries using Gamma correction
Normal	Modifies all LUT entries using a linear mapping with a positive slope
Reverse	Modifies all LUT entries using a linear mapping with a negative slope
Roll	Relocates LUT entries upwards or downwards as one block
Shift	Modifies all LUT entries using a logical shift
Slope	Modifies part of a LUT with a linear mapping
Threshold	Modifies all LUT entries using a threshold operation
Copy	Copies all LUT entries to another LUT resource
Load	Loads LUT entries from a file
Save	Saves LUT entries to a file
GetCapability	Gets the value of a low-level Sapera C library capability
GetParameter SetParameter	Gets/sets the value of a low-level Sapera C library parameter

SapLut | Base Class: SapManager | Basic Class Hierarchy Chart

Member Functions

The following functions are members of the SapLut Class.

SapLut::SapLut

```
SapLut(
  int numEntries = SapDefLutEntries
  SapFormat format = SapFormatUint8
  SapLocation loc = SapLocation::ServerSystem
);
SapLut(
  const char *filename,
  SapLocation loc = SapLocation::ServerSystem
);
```

Parameters

numEntries Number of LUT entries

format Data format for the LUT resource, can be one of the following values.

Monochrome (unsigned)	
SapFormatMono8	8-bit
SapFormatMono9	9-bit
...	...
SapFormatMono15	15-bit
SapFormatMono16	16-bit
Monochrome (unsigned)	
SapFormatInt8	8-bit
SapFormatInt9	9-bit
...	...
SapFormatInt15	15-bit
SapFormatInt16	16-bit
Color (non-interlaced)	
SapFormatColorNI8	8-bit
SapFormatColorNI9	9-bit
...	...
SapFormatColorNI15	15-bit
SapFormatColorNI16	16-bit
Color (interlaced)	
SapFormatColorI8	8-bit

	SapFormatColorI9	9-bit
	...	...
	SapFormatColorI15	15-bit
	SapFormatColorI16	16-bit

loc SapLocation object specifying the server where the LUT resource is located and the index of the resource on this server

filename String containing the name of a Sapera LUT file from which to extract the *numEntries* and *format* parameters.

Remarks

The SapLut constructor does not actually create the low-level Sapera resources. To do this, you must call the Create method.

For non-interlaced color formats, the red/green/blue components for one LUT element are stored separately:

RRR ... RRR	Red components of all elements
GGG ... GGG	Green components of all elements
BBB ... BBB	Blue components of all elements

For interlaced color formats, the red/green/blue components for one LUT element are stored together:

RGBRGBRGB	First three elements
...	...
RGBRGBRGB	Last three elements

The constructor is automatically called from the SapAcquisition Class so that acquisition lookup tables may be managed automatically. You just need to call the SapAcquisition::GetLut and SapAcquisition::ApplyLut methods, together with the functionality in this class, for all required LUT manipulations.

The SapView Class also manages display LUTs automatically in a similar way.

SapLut | SapLut Class Members | Basic Class Hierarchy Chart

SapLut::Create

virtual BOOL **Create**();

Return Value

Returns TRUE if the object was successfully created, FALSE otherwise

Remarks

Creates all the low-level Sapera resources needed by the LUT object.

SapLut | SapLut Class Members | Basic Class Hierarchy Chart

SapLut::Destroy

virtual BOOL **Destroy**();

Return Value

Returns TRUE if the object was successfully destroyed, FALSE otherwise

Remarks

Destroys all the low-level Sapera resources needed by the LUT object.

SapLut | SapLut Class Members | Basic Class Hierarchy Chart

SapLut::GetLocation

SapLocation **GetLocation**();

Remarks

Returns the location where the LUT resource is located. This usually corresponds to the system server, unless a specific server was specified in the SapLut constructor.

SapLut | SapLut Class Members | Basic Class Hierarchy Chart

SapLut::GetNumEntries, SapLut::SetNumEntries

int **GetNumEntries**();

virtual BOOL **SetNumEntries** (int *numEntries*);

Remarks

Gets/sets the number of LUT entries.

The initial value for this attribute is 256, unless a specific value was specified in the SapLut constructor.

You can only call SetNumEntries before the Create method.

SapLut | SapLut Class Members | Basic Class Hierarchy Chart

SapLut::GetFormat, SapLut::SetFormat

SapFormat **GetFormat**();

virtual void **SetFormat**(SapFormat *format*);

Remarks

Gets/sets the LUT data.

The initial value for this attribute is SapFormatMono8, unless a specific value was specified in the SapLut constructor.

You can only call SetFormat before the Create method. See the SapLut constructor for possible values for *format*.

SapLut | SapLut Class Members | Basic Class Hierarchy Chart

SapLut::GetElementSize

int **GetElementSize()**;

Remarks

Gets the number of bytes required to store a single LUT element.

The initial value for this attribute is 1. It is then set to the LUT element size value when calling the Create method.

SapLut | SapLut Class Members | Basic Class Hierarchy Chart

SapLut::GetNumPages

int **GetNumPages()**;

Remarks

Gets the number of color planes in the LUT.

The initial value for this attribute is 1. It is then set to the LUT number of pages value when calling the Create method.

This value is usually 1 if the LUT format is monochrome and 3 if it is color.

SapLut | SapLut Class Members | Basic Class Hierarchy Chart

SapLut::IsSigned

BOOL **IsSigned**();

Remarks

Returns TRUE if the LUT data format is signed, FALSE otherwise.

The initial value for this attribute is FALSE. It is then set to the LUT signed value when calling the Create method.

SapLut | SapLut Class Members | Basic Class Hierarchy Chart

SapLut::GetTotalSize

int **GetTotalSize(**);

Remarks

Gets the number total of bytes required to store the LUT data

The initial value for this attribute is 256. It is then set to the LUT size value when calling the Create method.

SapLut | SapLut Class Members | Basic Class Hierarchy Chart

SapLut::GetHandle

virtual CORHANDLE **GetHandle**();

Remarks

Gets the low-level Sapera handle of the LUT resource that you may then use from the low-level Sapera functionality. The handle is only valid after you call the Create method.

See the *Sapera LT Basic Modules Reference Manual* for details on low-level Sapera functionality.

SapLut | SapLut Class Members | Basic Class Hierarchy Chart

SapLut::Read

virtual BOOL **Read**(int *lutIndex*, SapData **pValue*);

virtual BOOL **Read**(int *offset*, void **pData*, int *size*);

Parameters

lutIndex	Index of LUT element to read, starting at 0
pValue	Pointer to a destination value object
offset	Byte offset to start reading from in the LUT.
pData	Destination memory area for LUT data
size	Number of bytes to read

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Use the first form of Read to read a single LUT element to either a SapDataMono or SapDataRGB object. You do not have to know the exact LUT data format and how it is stored in memory.

Use the second form of Read if you want to read raw LUT data directly to a memory area allocated in the application program. In this case, you also need to use the GetFormat, GetElementSize, and GetNumPages methods to find out the LUT data organization.

SapLut | SapLut Class Members | Basic Class Hierarchy Chart

SapLut::Write

virtual BOOL **Write**(int *lutIndex*, SapData *value*);

virtual BOOL **Write** (int *offset*, void **pData*, int *size*);

Parameters

lutIndex	Index of LUT element to write, starting at 0
value	Source value object
offset	Byte offset to start writing to in the LUT.
pData	Source memory area for LUT data
size	Number of bytes to write

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Use the first form of Write to write a single LUT element from either a SapDataMono or SapDataRGB object. You do not have to know how the LUT is stored in memory.

Use the second form of Write if you want to write raw LUT data directly from a memory area allocated in the application program. In this case, you also need to use the GetFormat, GetElementSize, and GetNumPages methods to find out the LUT data organization.

SapLut | SapLut Class Members | Basic Class Hierarchy Chart

SapLut::Arithmetic

virtual BOOL **Arithmetic**(SapLut::ArithmeticOp *operation*, SapData *value*);

Parameters

operation	Specifies how to modify LUT data elements. The following operations are available:	
	SapLut::Add	Addition with saturation: data[i] = min(maxValue, data[i] + *value*)
	SapLut::Asub	Absolute subtraction: data[i] = abs(data[i] – *value*)
	SapLut::Max	Maximum value: data[i] = max(data[i], *value*)
	SapLut::Min	Minimum value: data[i] = min(data[i], *value*)
	SapLut::Scale	Scale to smaller maximum value: data[i] = (data[i] * *value*) / maxValue
	SapLut::Sub	Subtraction with saturation: data[i] = max(minValue, data[i] – *value)*
value	Source value object	

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Modifies all LUT entries using an arithmetic operation. The *value* must be either a SapDataMono or SapDataRGB object, depending on the LUT format.

SapLut | SapLut Class Members | Basic Class Hierarchy Chart

SapLut::BinaryPattern

virtual BOOL **BinaryPattern**(int *bitNumber*, SapData *newValue*);

Parameters

bitNumber Bit number that identifies the indices of the LUT data elements to modify

newValue Source value object

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Modifies some LUT entries based on a binary pattern. Only the entries with indices that have the *bitNumber* bit set are modified using *newValue*. Each entry is calculated as follows:

data[i] = (i & (1 << *bitNumber*)) ? *newValue* : data[i]

The *value* must be either a SapDataMono or SapDataRGB object, depending on the LUT format.

SapLut | SapLut Class Members | Basic Class Hierarchy Chart

SapLut::Boolean

virtual BOOL **Boolean**(SapLut::BooleanOp *operation*, SapData *value*);

Parameters

operation Specifies how to modify LUT data elements. The following operations are available:

SapLut::And	Boolean AND: data[i] = data[i] & *value*
SapLut::Or	Boolean OR: data[i] = data[i] | *value*
SapLut::Xor	Boolean XOR: data[i] = data[i] ^ *value*

value Source value object

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Modifies all LUT entries using a Boolean operation. The *value* must be either a SapDataMono or SapDataRGB object, depending on the LUT format.

SapLut | SapLut Class Members | Basic Class Hierarchy Chart

SapLut::Gamma

virtual BOOL **Gamma**(float *factor*);

Parameters

factor Gamma correction factor to apply

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Modifies all LUT entries using inverse gamma correction with the specified *factor*. This is used to correct the light response of the camera, which is often a power function (referred to as the gamma function). A *factor* of 1 means no correction is applied, and a normal LUT is computed instead.

SapLut | SapLut Class Members | Basic Class Hierarchy Chart

SapLut::Normal

virtual BOOL **Normal**();

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Modifies all LUT entries using a linear mapping with a positive slope, as follows:

```
data[0] = minValue
(Linear mapping from data[0] to data[maxIndex])
data[maxIndex] = maxValue
```

SapLut | SapLut Class Members | Basic Class Hierarchy Chart

SapLut::Reverse

virtual BOOL **Reverse**();

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Modifies all LUT entries using a linear mapping with a negative slope, as follows:

data[0] = maxValue
(Linear mapping from data[0] to data[maxIndex])
data[maxIndex] = minValue

SapLut | SapLut Class Members | Basic Class Hierarchy Chart

SapLut::Roll

virtual BOOL **Roll**(int *numEntries*);

Parameters

numEntries Specifies by how many entries LUT data should be shifted

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Relocates LUT entries upwards or downwards as one block. The actual data elements are not modified, and their position relative to one another remains the same. If *numEntries* is positive, then a downward shift occurs. If it is negative, an upward shift occurs. This behavior is expressed as follows:

If *numEntries* > 0: data[(i + *numEntries*) % maxIndex] = data[i]
If *numEntries* < 0: data[i] = data[(i - *numEntries*) % maxIndex]

SapLut | SapLut Class Members | Basic Class Hierarchy Chart

SapLut::Shift

virtual BOOL **Shift**(int *numBits*);

Parameters

numBits Specifies by how many bits LUT entries should be shifted

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Modifies all LUT entries using a logical shift. If *numBits* is positive, a left shift occurs, and the least significant bits are filled with 0's. If *numBits* is negative, a right shift occurs, and the most significant bits are filled with 0's. This behavior is expressed as follows:

If *numBits* > 0: data[i] <<= *numBits*
If *numBits* < 0: data[i] >>= (-*numBits*)

SapLut | SapLut Class Members | Basic Class Hierarchy Chart

SapLut::Slope

virtual BOOL **Slope**(int *startIndex*, int *endIndex*, SapData *minValue*, SapData *maxValue*, BOOL *clipOutside* = FALSE);

Parameters

startIndex	Starting LUT index for linear mapping
endIndex	Ending LUT index for linear mapping
minValue	LUT element value at starting index
maxValue	LUT element value at ending index
clipOutside	Specifies whether LUT elements outside the mapping range should also be modified

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Modifies part of a LUT with a linear mapping. LUT elements from *startIndex* to *endIndex* are remapped from *minValue* to *maxValue*. If *clipOutside* is TRUE, then elements outside the range are unaffected. If FALSE, then elements below *startIndex* are set to *minValue* and elements above *endIndex* are set to *maxValue*. This behavior is expressed as follows:

If *clipOutside* is TRUE: data[0] ... data[*startIndex* - 1] = *minValue*
data[*startIndex*] = *minValue*
(Linear mapping from data[*startIndex*] to data[*endIndex*])
data[*endIndex*] = *maxValue*
If *clipOutside* is TRUE: data[*endIndex* + 1] ... data[maxIndex- 1] = *maxValue*

The value arguments must be either SapDataMono or SapDataRGB objects, depending on the LUT format.

SapLut | SapLut Class Members | Basic Class Hierarchy Chart

SapLut::Threshold

virtual BOOL **Threshold**(SapData *threshValue*);

virtual BOOL **Threshold**(SapData *lowValue*, SapData *highValue*);

Parameters

threshValue	Reference value for single threshold
lowValue	Lower reference value for double threshold
highValue	Upper reference value for double threshold

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Modifies all LUT elements using a threshold operation.

The first form of Threshold implements single threshold. Elements with a value lower than *threshValue* are set to the lowest possible value. Elements with a value higher than or equal to *threshValue* are set to the highest possible value. This behavior is expressed as follows:

data[i] = (data[i] >= *threshValue*) ? maxValue : minValue

The second form implements double threshold. Elements with a value higher than or equal to *lowValue*, but lower than *highValue*, are set to the highest possible value. Elements outside that range are set to the lowest possible value. This behavior is expressed as follows:

data[i] = (data[i] >= *lowValue* && data[i] < *highValue*) ? maxValue : minValue

The value arguments must be either SapDataMono or SapDataRGB objects, depending on the LUT format.

SapLut | SapLut Class Members | Basic Class Hierarchy Chart

SapLut::Copy

virtual BOOL **Copy**(SapLut **pSrc*);

Parameters

pSrc LUT object to copy from

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Copies all source LUT object entries to the current object. The two LUTs must be exactly the same size, as returned by the GetTotalSize method.

SapLut | SapLut Class Members | Basic Class Hierarchy Chart

SapLut::Load

virtual BOOL **Load**(const char **filename*);

Parameters

filename Name of source file

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Loads LUT entries from a file. The number of entries and formats of the LUT are updated to reflect the file contents. After calling Load, use the GetNumEntries and GetFormat methods to get their updated values.

SapLut | SapLut Class Members | Basic Class Hierarchy Chart

SapLut::Save

virtual BOOL **Save**(const char **filename*);

Parameters

filename Name of destination file

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Saves LUT entries to a file.

SapLut | SapLut Class Members | Basic Class Hierarchy Chart

SapLut::GetCapability

virtual BOOL **GetCapability**(int *cap*, void **pValue*);

Parameters

param	Low-level Sapera C library capability to read
pValue	Pointer to capability value to read back

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

This method allows direct read access to low-level Sapera C library capabilities for the LUT module. It needs a pointer to a memory area large enough to receive the capability value, which is usually a 32-bit integer.

You will rarely need to use GetCapability. The SapLut Class already uses important capabilities internally for self-configuration and validation.

See the *Sapera LT Basic Modules Reference Manual* for a description of all capabilities and their possible values.

SapLut | SapLut Class Members | Basic Class Hierarchy Chart

SapLut::GetParameter, SapLut::SetParameter

virtual BOOL **GetParameter**(int *param*, void **pValue*);

virtual BOOL **SetParameter**(int *param*, int *value*);

virtual BOOL **SetParameter**(int *param*, void **pValue*);

Parameters

param	Low-level Sapera C library parameter to read or write
pValue	Pointer to parameter value to read back or to write
value	New parameter value to write

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

These methods allow direct read/write access to low-level Sapera C library parameters for the LUT module. The GetParameter method needs a pointer to a memory area large enough to receive the parameter value, which is usually a 32-bit integer. The first form of SetParameter accepts a 32-bit value for the new value. The second form takes a pointer to the new value and is required when the parameter uses more than 32-bits of storage.

Note that you will rarely need to use these methods. You should first make certain that what you need is not already supported through the SapLut Class. Also, directly setting parameter values may interfere with the correct operation of the class.

See the *Sapera LT Basic Modules Reference Manual* for a description of all parameters and their possible values.

SapLut | SapLut Class Members | Basic Class Hierarchy Chart

SapManager

The SapManager Class includes methods for describing the Sapera resources present on the system. It also includes error management capabilities.

You will never need to explicitly create a SapManager object. First, almost all methods are declared as static, which means you may use them at any time. Second, most Sapera++ classes are derived from SapManager, so they inherit its methods and protected data members.

#include <SapClassBasic.h>

SapManager Class Members | Basic Class Hierarchy Chart

SapManager Class Members

Attributes	
operator BOOL	Checks whether the Create method has succeeded for a derived object
GetDisplayStatusMode, SetDisplayStatusMode	Gets/sets the global reporting mode for messages and errors
GetResetTimeout SetResetTimeout	Gets/sets the timeout value used when resetting a hardware device
GetPixelDepthMin, GetPixelDepthMax	Gets the minimum and maximum number of significant bits for a given data format
Operations	
GetServerCount	Gets the number of available Sapera servers
GetServerIndex	Gets the index of a Sapera server
GetServerName	Gets the name of a Sapera server
GetServerType	Gets the type of a Sapera server
IsServerAccessible	*Checks if the resources for a server are accessible*
GetServerHandle	Returns the low-level Sapera handle of a server resource
GetServerSerialNumber	Gets the serial number corresponding to a Sapera server

GetResourceCount	Gets the number of Sapera resources of a specific type on a server
GetResourceIndex	Gets the index of a Sapera resource
GetResourceName	Gets the name of a Sapera resource
IsResourceAvailable	Checks whether a resource is available for use
IsSystemLocation	Check whether a SapLocation object is located on the system server
IsSameServer	Checks whether two SapLocation objects are located on the same server
IsSameLocation	Checks whether two SapLocation objects are the same
GetFormatType	Gets the data type corresponding to a Sapera data format
GetStringFromFormat	Gets a text description of a Sapera data format
IsStatusOk	Checks the return value of a Sapera low-level C Library function call, and reports an error if appropriate
GetLastStatus	Gets a description of the latest Sapera low-level C library error
DisplayMessage	Reports a custom message using the current reporting mode
ResetServer	Resets the hardware device associated with a specific server
GetInstallDirectory	Gets the directory where a Sapera product is installed
RegisterServerCallback	Registers a callback function for server related events
UnregisterServerCallback	Unregister the callback function for server related events

SapManager | Basic Class Hierarchy Chart

Member Functions

The following functions are members of the SapManager Class.

SapManager::operator BOOL

operator BOOL();

Remarks

Checks whether the Create method has succeeded for a derived object, returning TRUE if successful, FALSE otherwise. This allows the variable representing the object to be used in a Boolean expression.

Calling the Destroy method resets this attribute to FALSE.

SapManager | SapManager Class Members | Basic Class Hierarchy Chart

SapManager::GetDisplayStatusMode, SapManager::SetDisplayStatusMode

static SapManager::StatusMode **GetDisplayStatusMode**();

static BOOL **SetDisplayStatusMode**(SapManager::StatusMode *mode,* SapManCallback *pCallback* = NULL, void **pContext* = NULL);

Parameters

mode	New reporting mode. The following values are available:	
	SapManager::StatusNotify	Sends messages to a popup window
	SapManager::StatusLog	Sends messages to the DALSA Log Viewer
	SapManager::StatusDebug	Sends messages to the active debugger, if any
	SapManager::StatusCustom	Messages are not sent anywhere, they are just stored internally
	SapManager::StatusCallback	Sends messages to application code through a callback function
pCallback	Application callback function to be called when reporting a message.	
	The callback function must be declared as:	
	void MyCallback(SapManCallbackInfo **pInfo*);	
pContext	Optional pointer to an application context to be passed to the callback function. If *pCallback* is NULL, this parameter is ignored.	

Remarks

Gets/sets the global reporting mode for messages and errors. The new mode is then used by the IsStatusOk and DisplayMessage methods.

For StatusCustom reporting mode, the only way to retrieve messages is by calling the GetLastStatus method.

For StatusCallback reporting mode, you can call the following SapManCallbackInfo methods from the application callback function to retrieve the relevant information: GetErrorValue, GetErrorMessage, and GetContext.

The initial value for this attribute is StatusNotify.

SapManager | SapManager Class Members | Basic Class Hierarchy Chart

SapManager::GetResetTimeout, SapManager::SetResetTimeout

static int **GetResetTimeout**();

static void **SetResetTimeout** (int *timeOut*);

Remarks

Gets/sets the timeout value (in milliseconds) used when resetting a hardware device. This value is used by the ResetServer method.

The initial value for this attribute is 2000, (20 seconds).

SapManager | SapManager Class Members | Basic Class Hierarchy Chart

SapManager::GetPixelDepthMin, SapManager::GetPixelDepthMax

static int **GetPixelDepthMin**(SapFormat *format*);

static int **GetPixelDepthMax**(SapFormat *format*);

Remarks

Gets the minimum and maximum number of significant bits for a given buffer *format*. This corresponds to the minimum and maximum pixel depth values for a corresponding SapBuffer object.

See SapBuffer::SapBuffer for a list of possible values for *format.*

SapManager | SapManager Class Members | Basic Class Hierarchy Chart

SapManager::GetServerCount

static int **GetServerCount**();

static int **GetServerCount**(SapManager::ResType *resourceType*);

Parameters

resourceType	Resource type to inquire, can be one of the following:	
	SapManager::ResourceAcq	Acquisition hardware
	SapManager::ResourceCab	DALSA Coreco Auxiliary Bus
	SapManager::ResourceCounter	Event counters
	SapManager::ResourceDisplay	Physical displays
	SapManager::ResourceDsp	Digital Signal Processors
	SapManager::ResourceGio	General inputs and outputs
	SapManager::ResourceGraphic	Graphics engine
	SapManager::ResourcePixPro	Pixel Processor hardware
	SapManager::ResourceRtPro	Realtime Processor hardware

Remarks

Gets the number of available Sapera servers.

The first form of this method considers all servers, regardless of their resource type. In this case, the return value is at least 1, since the system server is always present.

The second form returns the number of servers for the specified resource type only, so the return value may be equal to 0.

SapManager | SapManager Class Members | Basic Class Hierarchy Chart

SapManager::GetServerIndex

static int **GetServerIndex**(const char **serverName*);

static int **GetServerIndex**(SapLocation *loc*);

Parameters

serverName	Sapera server name
loc	Valid SapLocation object

Remarks

Gets the index of a Sapera server. Returns SapLocation::ServerUnknown if the specified server cannot be found.

The first form of GetServerIndex uses the server name. The second form uses an existing SapLocation object with valid server information.

Use the Sapera Configuration utility to find the names of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names for that product.

SapManager | SapManager Class Members | Basic Class Hierarchy Chart

SapManager::GetServerName

static BOOL **GetServerName**(int *serverIndex*, char **serverName*);

static BOOL **GetServerName**(SapLocation *loc*, char **serverName*);

static BOOL **GetServerName**(int *serverIndex*, SapManager::ResType *resourceType*, char **serverName*);

Parameters

serverIndex	Sapera server index
serverName	Memory area large enough to receive the server name (at least 32 bytes)
loc	Valid SapLocation object
resourceType	Resource type to inquire. See the SapManager::GetServerCount method for the list of possible values.

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the name of a Sapera server.

The first form of GetServerName uses a server index between 0 and the value returned by the GetServerCount method, minus 1.

The second form uses an existing SapLocation object with valid server information.

The third form only considers servers with at least one resource of the specified type. For example, index 1 corresponds to the second server with at least one acquisition device.

Use the Sapera Configuration utility to find the names of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names for that product.

SapManager | SapManager Class Members | Basic Class Hierarchy Chart

SapManager::GetServerType

static SapManager::ServerType **GetServerType**(int *serverIndex*);

static SapManager::ServerType **GetServerType**(const char **serverName*);

static SapManager::ServerType **GetServerType**(SapLocation *loc*);

Parameters

serverIndex	Sapera server index
serverName	Sapera server name
loc	Valid SapLocation object

Return Value Can be one of the following:

SapManager::ServerNone	Server type cannot be determined
SapManager::ServerSystem	System server
SapManager::ServerViperRGB	Viper-RGB acquisition board
SapManager::ServerViperDigital	Viper-Digital acquisition board
SapManager::ServerViperQuad	Viper-Quad acquisition board
SapManager::ServerViperCamLink	Viper-CamLink acquisition board
SapManager::ServerBanditII	Bandit II VGA frame grabber
SapManager::ServerX64CL	X64-CL Camera Link acquisition board
SapManager::ServerX64CLiPRO	X64-CL iPro Camera Link acquisition board
SapManager::ServerX64LVDS	X64-LVDS acquisition board
SapManager::ServerX64NS	NetSightII-CL acquisition board
SapManager::ServerX64ANQuad	X64-AN Quad acquisition board
SapManager::ServerPC2Vision	PC2-Vision acquisition board
SapManager::ServerPC2CamLink	PC2-CamLink acquisition board
SapManager::ServerAnacondaCL	Anaconda-CL vision processor
SapManager::ServerAnacondaLVDS	Anaconda-LVDS vision processor

Remarks

Gets the type of a Sapera server.

The first form of GetServerType uses a server index between 0 and the value returned by the GetServerCount method, minus 1. The second form uses a server name. The third form uses an existing SapLocation object with valid server information.

Use the Sapera Configuration utility to find the names of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names for that product.

SapManager | SapManager Class Members | Basic Class Hierarchy Chart

SapManager::IsServerAccessible

static BOOL **IsServerAccessible**(int *serverIndex*);

static BOOL **IsServerAccessible**(const char **serverName*);

static BOOL **IsServerAccessible**(SapLocation *loc*);

Parameters

serverIndex	Index of Sapera server containing the resource
serverName	Name of Sapera server containing the resource
loc	Valid SapLocation object

Return Value

Returns TRUE if the resources for the server are accessible, FALSE otherwise

Remarks

Checks if the resources belonging to a server are currently accessible. Although existing objects for these resources are still valid when their server becomes unaccessible, they must be left alone or destroyed (for example, SapAcqDevice::Destroy).

When a Sapera application starts, all detected servers are automatically accessible. However, some Sapera devices like GigE cameras can be connected and disconnected while a Sapera application is running. When such a device is connected for the first time, its server is automatically accessible. When the device is later disconnected, the server becomes unaccessible. If it is reconnected again, the server is once again accessible.

The first form of IsServerAccessible uses a server index. Specify a server index between 0 and the value returned by the GetServerCount method, minus 1. The second form uses a server name. The third form uses an existing SapLocation object.

Accessibility of servers can also be determined by registering callbacks for server related events using the RegisterServerCallback method..

SapManager | SapManager Class Members | Basic Class Hierarchy Chart

SapManager::GetServerHandle

static BOOL **GetServerHandle**(int *serverIndex*, PCORSERVER *pServer*);

static BOOL **GetServerHandle**(const char **serverName*, PCORSERVER *pServer*);

static BOOL **GetServerHandle**(SapLocation *loc*, PCORSERVER *pServer*);

Parameters

serverIndex	Sapera server index
serverName	Sapera server name
loc	Valid SapLocation object
pServer	Pointer to returned low-level Sapera server handle

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the low-level handle of a Sapera server.

The first form of GetServerHandle uses a server index between 0 and the value returned by the GetServerCount method, minus 1. The second form uses a server name. The third form uses an existing SapLocation object with valid server information.

Use the Sapera Configuration utility to find the names of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names for that product.

See the *Sapera LT Basic Modules Reference Manual* for details on low-level Sapera functionality.

SapManager | SapManager Class Members | Basic Class Hierarchy Chart

SapManager::GetServerSerialNumber

static BOOL **GetServerSerialNumber**(int *serverIndex*, char **serialNumber*);

static BOOL **GetServerSerialNumber**(const char **serverName*, char **serialNumber*);

static BOOL **GetServerSerialNumber**(SapLocation *loc*, char **serialNumber*);

Parameters

serverIndex	Sapera server index
serialNumber	Memory area large enough to receive the text for the serial number (at least 16 bytes)
serverName	Sapera server name
loc	Valid SapLocation object

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets a text representation of the serial number corresponding to the hardware device for the specified Sapera server. It consists of either the letter 'S' or 'H' followed by seven digits, for example, "S1234567".

SapManager | SapManager Class Members | Basic Class Hierarchy Chart

SapManager::GetResourceCount

static int **GetResourceCount**(int *serverIndex*, SapManager::ResType *resourceType*);

static int **GetResourceCount**(const char **serverName*, SapManager::ResType *resourceType*);

static int **GetResourceCount**(SapLocation *loc*, SapManager::ResType *resourceType*);

Parameters

serverIndex	Sapera server index
resourceType	Resource type to inquire. See the SapManager::GetServerCount method for the list of possible values.
serverName	Sapera server name
loc	Valid SapLocation object

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the number of resources of a specified type on a Sapera server. This only applies to static resources, that is, those attached to physical devices. Dynamic resources, like buffers, do not have a fixed count.

The first form of GetResourceCount uses a server index between 0 and the value returned by the GetServerCount method, minus 1. The second form uses a server name. The third form uses an existing SapLocation object with valid server information.

Use the Sapera Configuration utility to find the names of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names for that product.

SapManager | SapManager Class Members | Basic Class Hierarchy Chart

SapManager::GetResourceIndex

static int **GetResourceIndex**(int *serverIndex*, SapManager::ResType *resourceType*, const char **resourceName*);

static int **GetResourceIndex**(const char **serverName*, SapManager::ResType *resourceType*, const char **resourceName*);

Parameters

serverIndex	Sapera server index
resourceType	Resource type to inquire. See the SapManager::GetServerCount method for the list of possible values.
resourceName	Sapera resource name
serverName	Sapera server name

Remarks

Gets the index of a Sapera resource. Returns SapLocation::ResourceUnknown if the specified resource cannot be found.

The first form of GetResourceIndex looks for the resource of the specified name and type on the server specified by *index*.

The second form uses the server name instead of the index.

Use the Sapera Configuration utility to find the names of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server and resource names for that product.

SapManager | SapManager Class Members | Basic Class Hierarchy Chart

SapManager::GetResourceName

static BOOL **GetResourceName**(int *serverIndex*, SapManager::ResType *resourceType*, int *resourceIndex*, char **resourceName*);

static BOOL **GetResourceName**(const char **serverName*, SapManager::ResType *resourceType*, int *resourceIndex*, char * *resourceName*);

static BOOL **GetResourceName**(SapLocation *loc*, SapManager::ResType *resourceType*, char * *resourceName*);

Parameters

serverIndex	Index of Sapera server containing the resource
resourceType	Resource type to inquire. See the SapManager::GetServerCount method for the list of possible values.

resourceIndex	Index of requested resource of the specified type
resourceName	Memory area large enough to receive the resource name (at least 128 bytes)
serverName	Name of Sapera server containing the resource
loc	Valid SapLocation object

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the name of a Sapera resource of a specified type.

The first form of GetResourceName uses server and resource indices. Specify a server index between 0 and the value returned by the GetServerCount method, minus 1. Specify a resource index between 0 and the value returned by the GetResourceCount method, minus 1.

The second form uses a server name and resource index. The third form uses an existing SapLocation object with valid server and resource information.

Use the Sapera Configuration utility to find the names of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names and resource indices for that product.

SapManager | SapManager Class Members | Basic Class Hierarchy Chart

SapManager::IsResourceAvailable

static BOOL **IsResourceAvailable**(int *serverIndex*, SapManager::ResType *resourceType*, int *resourceIndex*);

static BOOL **IsResourceAvailable**(const char **serverName*, SapManager::ResType *resourceType*, int *resourceIndex*);

static BOOL **IsResourceAvailable** (SapLocation *loc*, SapManager::ResType *resourceType*);

Parameters

serverIndex	Index of Sapera server containing the resource
resourceType	Resource type to inquire. See the SapManager::GetServerCount method for the list of possible values.
resourceIndex	Index of requested resource of the specified type
serverName	Name of Sapera server containing the resource
loc	Valid SapLocation object

Return Value

Returns TRUE if the specified resource is not already used, FALSE otherwise

Remarks

Determines if a specific Sapera resource on a server is available. You may use IsResourceAvailable, for example, before calling SapAcquisition::Create to avoid getting an error when the acquisition resource is already in use.

The first form of IsResourceAvailable uses server and resource indices. Specify a server index between 0 and the value returned by the GetServerCount method, minus 1. Specify a resource index between 0 and the value returned by the GetResourceCount method, minus 1.

The second form uses a server name and resource index. The third form uses an existing SapLocation object with valid server and resource information.

Use the Sapera Configuration utility to find the names of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names and resource indices for that product.

SapManager | SapManager Class Members | Basic Class Hierarchy Chart

SapManager::IsSystemLocation

static BOOL **IsSystemLocation**();

static BOOL **IsSystemLocation**(SapLocation *loc*);

Parameters

loc Valid SapLocation object

Return Value

Returns TRUE if the current application is running on the system server, or if the SapLocation object refers to the system server, FALSE otherwise

SapManager | SapManager Class Members | Basic Class Hierarchy Chart

SapManager::IsSameServer

static BOOL **IsSameServer**(SapLocation *loc1*, SapLocation *loc2*);

Parameters

loc1 First valid SapLocation object

loc2 Second valid SapLocation object

Return Value

Returns TRUE if the two specified SapLocation objects have the same server information, FALSE otherwise

SapManager | SapManager Class Members | Basic Class Hierarchy Chart

SapManager::IsSameLocation

static BOOL **IsSameLocation**(SapLocation *loc1*, SapLocation *loc2*);

Parameters

loc1 First valid SapLocation object

loc2 Second valid SapLocation object

Return Value

Returns TRUE if the two specified SapLocation objects have the same server and resource information, FALSE otherwise

SapManager | SapManager Class Members | Basic Class Hierarchy Chart

SapManager::GetFormatType

static SapFormatType **GetFormatType**(SapFormat *format*);

Parameters

format Sapera data format

Remarks

Gets the data type corresponding to the specified Sapera data format as one of the following values:

SapFormatTypeUnknown	Unable to determine data type
SapFormatTypeMono	Monochrome
SapFormatTypeRGB	RGB color
SapFormatTypeYUV	YUV color
SapFormatTypeHSI	HSI color
SapFormatTypeHSV	HSV color
SapFormatTypeFloat	Signed floating point
SapFormatTypePoint	Signed integer X and Y components
SapFormatTypeFPoint	Signed floating-point X and Y components
SapFormatTypeColor	Lookup table color data
SapFormatTypeRGBA	RGB color with an additional component (alpha channel, infrared component, etc.)

SapManager | SapManager Class Members | Basic Class Hierarchy Chart

SapManager::GetStringFromFormat

static BOOL **GetStringFromFormat**(SapFormat *format*, char * *txtFormat*);

Parameters

format	Sapera data format
txtFormat	Memory area large enough to receive the description (at least 16 bytes)

Return Value

Returns TRUE if the low-level function call succeeded, FALSE otherwise

Remarks

Gets a text description of the specified Sapera data format, for example, 'MONO8' for SaprFormatMono8.

SapManager | SapManager Class Members | Basic Class Hierarchy Chart

SapManager::IsStatusOk

static BOOL **IsStatusOk**(const char **functionName*, SAPSTATUS *status*);

Parameters

functionName	Name of a low-level Sapera function
status	Low-level status code returned by the function

Return Value

Returns TRUE if the low-level function call succeeded, FALSE otherwise

Remarks

Checks the return value of a Sapera low-level C library function call and reports an error if appropriate, using the current reporting mode. See the SetDisplayStatusMode method for a description of the available modes.

See the *Sapera LT Basic Modules Reference Manual* for details on low-level Sapera functionality.

SapManager | SapManager Class Members | Basic Class Hierarchy Chart

SapManager::GetLastStatus

static const char ***GetLastStatus**();

static void **GetLastStatus**(SAPSTATUS **pLastStatus*);

Parameters

pLastStatus	Pointer to the most recent Sapera low-level status code to retrieve

Remarks

Gets a description of the latest Sapera++ and/or low-level C library error.

The first form of GetLastStatus returns the latest text description, similar to what is generated by the IsStatusOk method. If the actual error occurred inside a call to the low-level C library, then you may also use the second form to retrieve the actual error code.

See the SetDisplayStatusMode method for a description of the available reporting modes.

See the *Sapera LT Basic Modules Reference Manual* for details on low-level Sapera functionality.

SapManager | SapManager Class Members | Basic Class Hierarchy Chart

SapManager::DisplayMessage

static void **DisplayMessage**(const char **message*, const char **fileName* = NULL, int *lineNumber* = 0, ...);

Parameters

message	Custom message to report
fileName	Name of source file from which DisplayMessage is called
lineNumber	Line number from which DisplayMessage is called
...	Variable arguments if *message* includes printf-style format specifications

Remarks

Reports a custom *message* using the current reporting mode. File and line information is automatically appended to *message*, unless you set *fileName* to NULL.

See the SetDisplayStatusMode method for a description of the available reporting modes.

SapManager | SapManager Class Members | Basic Class Hierarchy Chart

SapManager::ResetServer

static BOOL **ResetServer**(int *serverIndex*, BOOL *wait* = TRUE, SapManCallback *pCallback* = NULL, void **pContext* = NULL);

static BOOL **ResetServer**(const char **serverName*, BOOL *wait* = TRUE, SapManCallback *pCallback* = NULL, void **pContext* = NULL);

static BOOL **ResetServer**(SapLocation *loc*, BOOL *wait* = TRUE, SapManCallback *pCallback* = NULL, void **pContext* = NULL);

Parameters

serverIndex	Sapera server index
serverName	Sapera server name
loc	Valid SapLocation object
wait	Specifies whether ResetServer should return immediately after resetting the specified server, or if it should wait for the server to be operational again
pCallback	Application callback function to be called when the server is operational again after a reset. The callback function must be declared as: void MyCallback(SapManCallbackInfo **pInfo*);
pContext	Optional pointer to an application context to be passed to the callback function. If *pCallback* is NULL, this parameter is ignored.

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Resets the hardware device associated with a specific server.

The first form of ResetServer uses a server index between 0 and the value returned by the GetServerCount method, minus 1. The second form uses a server name. The third form uses an existing SapLocation object with valid server information.

There are three ways to use ResetServer:

wait = TRUE *pCallback* = don't care	Returns only when the reset is complete, and the server is operational again
wait = FALSE *pCallback* = NULL	Returns immediately after resetting the server. The application program is then responsible for figuring out when the server is operational again.
wait = FALSE *pCallback* != NULL	Returns immediately after resetting the server, and notifies the application using the callback function when the server is operational again

You can call the following SapManCallbackInfo methods from the application callback function to retrieve the

relevant information: GetServerIndex and GetContext.

Use the Sapera Configuration utility to find the names of all Sapera servers in your system.

See also the 'Servers and Resources' section in the user's manual for each Sapera hardware product for a list of all valid server names for that product.

SapManager | SapManager Class Members | Basic Class Hierarchy Chart

SapManager::GetInstallDirectory

static BOOL **GetInstallDirectory**(int *serverIndex*, char **installDir*);

static BOOL **GetInstallDirectory** (const char **serverName*, char **installDir*);

static BOOL **GetInstallDirectory** (SapLocation *loc*, char **installDir*);

Parameters

serverIndex	Sapera server index
installDir	Memory area large enough to receive the installation directory (at least 257 bytes)
serverName	Sapera server name
loc	Valid SapLocation object

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the directory where a Sapera product is installed.

For the system server, this corresponds to the Sapera installation directory, for example, **c:\DALSA Coreco\Sapera**.

For a server corresponding to a hardware device, this corresponds to the directory where the driver for the device is installed, .for example, **c:\DALSA Coreco\BanditII**.

SapManager | SapManager Class Members | Basic Class Hierarchy Chart

SapManager::RegisterServerCallback

static BOOL **RegisterServerCallback**(SapManager::EventType *eventType*, SapManCallback c*allback,* void *c*ontext* = NULL);

Parameters

eventType	Manager events for which the application callback function will be called. One or more of the following values may be combined together using a bitwise OR operation:	
	SapManager::EventServerNew	A new device is connected while a Sapera application is already running
	SapManager::EventServerNotAccessible	The device corresponding to an existing server is disconnected
	SapManager::EventServerAccessible	The device corresponding to an existing, unaccessible server is reconnected
	SapManager::EventServerDataBaseFull	There is no room in the Sapera server database for a new device that has just been connected
	SapManager::EventResourceInfoChanged	The information describing a resource (typically its label) has changed
c*allback*	Application callback function to be called each time one of the events specified above is received.	
	The callback function must be declared as:	
	void MyCallback(SapManCallbackInfo *_pInfo_);	
c*ontext*	Optional pointer to an application context to be passed to the callback function.	

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Registers a callback function for server related events. The callback function provides information on the corresponding event (through the SapManCallbackInfo object). The context pointer is also returned by the callback function allowing you to exchange user information between the callback and your application context.

In the callback function, you can obtain the event type that triggered the callback by calling the SapManCallbackInfo::GetEventType method. For all events except EventServerDataBaseFull, you can obtain the index of the server by calling SapManCallbackInfo::GetServerIndex. For the EventResourceInfoChanged event, you can obtain the index of the affected resource by calling SapManCallbackInfo::GetResourceIndex.

Note that server related events are only available when dealing with Sapera devices like GigE cameras, that can be connected and disconnected while a Sapera application is running.

SapManager | SapManager Class Members | Basic Class Hierarchy Chart

SapManager::UnregisterServerCallback

static BOOL **UnregisterServerCallback**(void);

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Unregisters the callback function for server related events.

SapManager | SapManager Class Members | Basic Class Hierarchy Chart

SapManCallbackInfo

The SapManCallbackInfo Class acts as a container for storing all arguments to the callback function for the SapManager Class.

#include <SapClassBasic.h>

SapManCallbackInfo Class Members | Basic Class Hierarchy Chart

SapManCallbackInfo Class Members

Construction	
SapManCallbackInfo	Class constructor
Attributes	
GetEventType	Gets the manager events that triggered the call to the application callback
GetServerIndex	Gets the Sapera server index associated with the call to the application callback
GetResourceIndex	Gets the Sapera resource index associated with the call to the application callback
GetContext	Gets the application context associated with the callback
GetErrorValue	Gets the low-level Sapera C library error code associated with the callback
GetErrorMessage	Gets the error message associated with the callback

SapManCallbackInfo | Basic Class Hierarchy Chart

Member Functions

The following functions are members of the SapManCallbackInfo Class.

SapManCallbackInfo::SapManCallbackInfo

SapManCallbackInfo(
  int *serverIndex*,
  void **context*);

SapManCallbackInfo(
  SapManager::EventType eventType,
  int *serverIndex*,
  void **context*);

SapManCallbackInfo(
  SapManager::EventType eventType,
  int *serverIndex*,
  int *resourceIndex*,
  void **context*);

SapManCallbackInfo(
  SAPSTATUS *errorValue*,
  const char **errorMessage*,
  void **context*);

Parameters

eventType	Combination of manager events. See the SapManager::RegisterServerCallback method for the list of possible values.
serverIndex	Sapera server index
resourceIndex	Sapera resource index
context	Pointer to the application context
errorValue	Low-level Sapera C library error code
errorMessage	Error message as a text string

Remarks

SapManager objects create an instance of this class before each call to the application callback method, in order to combine all function arguments into one container.

SapManager uses this class in various situations. The first occurs when a server is operational again after being reset. The second case corresponds to the callback reporting mode set using the SapManager::GetDisplayStatusMode, SapManager::SetDisplayStatusMode method.

It is also possible to explicitly register a callback function using the SapManager::RegisterServerCallback method.

SapManCallbackInfo | SapManCallbackInfo Class Members | Basic Class Hierarchy Chart

SapManCallbackInfo::GetEventType

SapManager::EventType **GetEventType**();

Remarks

Gets the combination of manager events that triggered the call to the application callback. See the SapManager::RegisterServerCallback method for the list of possible values.

SapManCallbackInfo | SapManCallbackInfo Class Members | Basic Class Hierarchy Chart

SapManCallbackInfo::GetServerIndex

int **GetServerIndex**();

Remarks

Gets the Sapera server index associated with the call to the application callback

SapManCallbackInfo | SapManCallbackInfo Class Members | Basic Class Hierarchy Chart

SapManCallbackInfo::GetResourceIndex

int **GetResourceIndex**();

Remarks

Gets the Sapera resource index associated with the call to the application callback

SapManCallbackInfo | SapManCallbackInfo Class Members | Basic Class Hierarchy Chart

SapManCallbackInfo::GetContext

void ***GetContext**();

Remarks

Gets the application context associated with the call to the application callback

SapManCallbackInfo | SapManCallbackInfo Class Members | Basic Class Hierarchy Chart

SapManCallbackInfo::GetErrorValue

SAPSTATUS **GetErrorValue**();

Remarks

Gets the low-level Sapera C library error code associated with the call to the application callback.

See the *Sapera LT Basic Modules Reference Manual* for details on low-level Sapera functionality.

SapManCallbackInfo | SapManCallbackInfo Class Members | Basic Class Hierarchy Chart

SapManCallbackInfo::GetErrorMessage

const char ***GetErrorMessage**();

Remarks

Gets the error message associated with the call to the application callback as a text string.

SapManCallbackInfo | SapManCallbackInfo Class Members | Basic Class Hierarchy Chart

SapPerformance

The SapPerformance Class implements basic benchmarking functionality. It is used by the SapProcessing Class to evaluate the time it takes to process one buffer. You may also use it for your own benchmarking needs.

#include <SapClassBasic.h>

SapPerformance Class Members | Basic Class Hierarchy Chart

SapPerformance Class Members

Construction	
SapPerformance	Class constructor
Operations	
Reset	Resets the internal timer
GetTime	Gets the number of seconds elapsed since the last timer reset
GetTimeMilli	Gets the number of milliseconds elapsed since the last timer reset
GetTimeMicro	Gets the number of microseconds elapsed since the last timer reset

SapPerformance | Basic Class Hierarchy Chart

Member Functions

The following functions are members of the SapPerformance Class.

SapPerformance::SapPerformance

SapPerformance();

Remarks

The SapPerformance constructor initializes the internal timer and resets it

SapPerformance | SapPerformance Class Members | Basic Class Hierarchy Chartt

SapPerformance::Reset

void **Reset**();

Remarks

Resets the internal timer. Calling the GetTime, GetTimeMilli, or GetTimeMicro methods then returns the amount of time elapsed since the reset.

SapPerformance | SapPerformance Class Members | Basic Class Hierarchy Chart

SapPerformance::GetTime

float **GetTime**(BOOL *bReset*);

Parameters

bReset Specifies whether the internal timer should be reset after it has been queried

Remarks

Gets the number of seconds elapsed since the last timer reset

SapPerformance | SapPerformance Class Members | Basic Class Hierarchy Chart

SapPerformance::GetTimeMilli

float **GetTimeMilli**(BOOL *bReset*);

Parameters

bReset Specifies whether the internal timer should be reset after it has been queried

Remarks

Gets the number of milliseconds elapsed since the last timer reset

SapPerformance | SapPerformance Class Members | Basic Class Hierarchy Chart

SapPerformance::GetTimeMicro

float **GetTimeMicro**(BOOL *bReset*);

Parameters

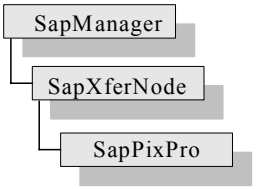
bReset Specifies whether the internal timer should be reset after it has been queried

Remarks

Gets the number of microseconds elapsed since the last timer reset

SapPerformance | SapPerformance Class Members | Basic Class Hierarchy Chart

SapPixPro



The SapPixPro Class includes the functionality to manipulate a pixel processor resource. It is used as an intermediate transfer node to allow data transfers from an acquisition resource to another transfer node (such as a buffer or CAB).

#include <SapClassBasic.h>

SapPixPro Class Members | Base Class: SapManager | Basic Class Hierarchy Chart

SapPixPro Class Members

Construction	
SapPixPro	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Attributes	
GetFile, SetFile	Gets/sets the name of the pixel processor file
GetParams, SetParams	Gets/sets the Pixel Processor parameters
GetOutputXferParams, SetOutputXferParams	Gets/sets the output transfer parameters
Operations	
ReadDesign	Reads a design file to allow retrieving information on functions and configurations
GetNumFunctions	Gets the number of standard functions available in the current design file
GetFunctionInfo	Gets information on a specific function in the current design file
GetNumConfigs	Gets the number of configurations available in the current design file
GetConfigInfo	Gets information on a specific configuration in the current design file

GetCapability	Gets the value of a low-level Sapera C library capability
GetParameter SetParameter	Gets/sets the value of a low-level Sapera C library parameter

SapPixPro | Base Class: SapManager | Basic Class Hierarchy Chart

Member Functions

The following functions are members of the SapPixPro Class.

SapPixPro::SapPixPro

```
SapPixPro(
  SapLocation loc = SapLocation::ServerSystem,
  const char *fileName = "",
  SapXferNode *pSrcNode = NULL,
  SapPixProParams *pParams = NULL
);
SapPixPro(
  SapLocation loc,
  const char * fileName,
  SapXferParams xferParams,
  SapPixProParams *pParams = NULL
);
```

Parameters

loc	SapLocation object specifying the server where the pixel processor resource is located and the index of this resource on the server
fileName	Name of the pixel processor file
pSrcNode	Source node object. To ensure transfer compatibility, this object must match the source node specified when adding a transfer pair (SapXferPair) to the SapTransfer object.
pParams	SapPixProParams object used to initialize function and configuration parameters
xferParams	Transfer parameters of the source node. To ensure transfer compatibility, its attributes must match the equivalent attributes of the source node specified when adding a transfer pair (SapXferPair) to the SapTransfer object.

Remarks

The SapPixPro constructor does not actually create the low-level Sapera resources. To do this, you must call the Create method.

Use the SapPixPro object only for loading the pixel processor file and configuring the parameters. To transfer

data through the Pixel Processor, you must use the SapTransfer Class (or one of its derived classes) and specify the SapPixPro object as a parameter. The data transfer is then controlled by the SapTransfer Class.

SapPixPro | SapPixPro Class Members | Basic Class Hierarchy Chart

SapPixPro::Create

virtual BOOL **Create**();

Return Value

Returns TRUE if the object was successfully created, FALSE otherwise

Remarks

Creates all the low-level Sapera resources needed by the pixel processor object and loads the design file into the Pixel Processor. Always call this method before SapTransfer::Create.

SapPixPro | SapPixPro Class Members | Basic Class Hierarchy Chart

SapPixPro::Destroy

virtual BOOL **Destroy**();

Return Value

Returns TRUE if the object was successfully destroyed, FALSE otherwise

Remarks

Destroys all the low-level Sapera resources needed by the pixel processor object. Always call this method after SapTransfer::Destroy.

SapPixPro | SapPixPro Class Members | Basic Class Hierarchy Chart

SapPixPro::GetFile, SapPixPro::SetFile

const char ***GetFile**();

virtual BOOL **SetFile**(const char **fileName*);

Remarks

Gets/sets the name of the pixel processor file.

You normally set the initial value for this attribute within the SapPixPro constructor. If you do not specify a value at that time, it then defaults to an empty string.

You can only call SetFile before the Create method.

SapPixPro | SapPixPro Class Members | Basic Class Hierarchy Chart

SapPixPro::GetParams, SapPixPro::SetParams

const SapPixProParams ***GetParams**();

virtual BOOL **SetParams**(SapPixProParams **pParams*);

Remarks

Gets/sets the pixel processor parameters.

You may set the initial value for this attribute in the SapPixPro constructor.

You can only call SetParams before the Create method.

SapPixPro | SapPixPro Class Members | Basic Class Hierarchy Chart

SapPixPro::GetOutputXferParams, SapPixPro::SetOutputXferParams

SapXferParams **GetOutputXferParams**();

virtual BOOL **SetOutputXferParams**(SapXferParams *params*);

Remarks

Gets/sets the output transfer parameters. Its attributes must match the equivalent attributes of the source node specified when adding a transfer pair (SapXferPair) to the SapTransfer object.

You can only call SetOutputXferParams before the Create method.

SapPixPro | SapPixPro Class Members | Basic Class Hierarchy Chart

SapPixPro::ReadDesign

BOOL **ReadDesign**(char **fileName*);

Parameters

fileName Name of the pixel processor file

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Reads a design file to allow retrieving information on functions and configurations. Use the GetNumFunctions, GetFunctionInfo**,** GetNumConfigs, and GetConfigInfo methods to obtain information on functions and configurations available in the file.

SapPixPro | SapPixPro Class Members | Basic Class Hierarchy Chart

SapPixPro::GetNumFunctions

int **GetNumFunctions**();

Remarks

Gets the number of standard functions available in the current design file. This information is only available after a design file has been loaded into the Pixel Processor.

SapPixPro | SapPixPro Class Members | Basic Class Hierarchy Chart

SapPixPro::GetFunctionInfo

BOOL **GetFunctionInfo**(int *index*, PCORPPL_FCT_PROP *pFuncProp*, char **funcName*);

Parameters

index	Function index between 0 and the value returned by the GetNumFunctions method, minus 1
pFuncProp	Pointer to a CORPPL_FCT_PROP structure for receiving the properties of the function
funcName	Memory area large enough to receive the function name (at least 64 bytes)

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets information on a specific function in the current design file. This information is only available after a design file has been loaded into the Pixel Processor.

The CORPPL_FCT_PROP structure contains an availability flag and memory bank ID. It is defined as follows:

```
typedef struct
{
  BOOLEAN fAvailable; // TRUE if function is available
  UINT32  bankId;     // Memory Bank where operation can be performed
} CORPPL_FCT_PROP, *PCORPPL_FCT_PROP;
```

SapPixPro | SapPixPro Class Members | Basic Class Hierarchy Chart

SapPixPro::GetNumConfigs

int **GetNumConfigs**();

Return Value

Gets the number of configurations available in the current design file. This information is only available after a design file has been loaded into the Pixel Processor.

SapPixPro | SapPixPro Class Members | Basic Class Hierarchy Chart

SapPixPro::GetConfigInfo

BOOL **GetConfigInfo**(int *index*, char **configName*);

Parameters

index	Configuration index between 0 and the value returned by the GetNumConfigs method, minus 1
configName	Memory area large enough to receive the configuration name (at least 64 bytes)

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets information on a specific configuration in the current design file. This information is only available after a design file has been loaded into the Pixel Processor.

SapPixPro | SapPixPro Class Members | Basic Class Hierarchy Chart

SapPixPro::GetCapability

virtual BOOL **GetCapability**(int *cap*, void **pValue*);

Parameters

param	Low-level Sapera C library capability to read
pValue	Pointer to capability value to read back

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

This method allows direct read access to low-level Sapera C library capabilities for the Pixel Processor module. It needs a pointer to a memory area large enough to receive the capability value, which is usually a 32-bit integer.

You will rarely need to use GetCapability. The SapPixPro Class already uses important capabilities internally for self-configuration and validation.

See the *Sapera LT Basic Modules Reference Manual* for a description of all capabilities and their possible values.

SapPixPro | SapPixPro Class Members | Basic Class Hierarchy Chart

SapPixPro::GetParameter, SapPixPro::SetParameter

virtual BOOL **GetParameter**(int *param*, void **pValue*);

virtual BOOL **SetParameter**(int *param*, int *value*);

virtual BOOL **SetParameter**(int *param*, void **pValue*);

Parameters

param	Low-level Sapera C library parameter to read or write
pValue	Pointer to parameter value to read back or to write
value	New parameter value to write

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

These methods allow direct read/write access to low-level Sapera C library parameters for the Pixel Processor module. The GetParameter method needs a pointer to a memory area large enough to receive the parameter value, which is usually a 32-bit integer. The first form of SetParameter accepts a 32-bit value for the new value. The second form takes a pointer to the new value and is required when the parameter uses more than 32-bits of storage.

Note that you will rarely need to use these methods. You should first make certain that what you need is not already supported through the SapPixPro Class. Also, directly setting parameter values may interfere with the correct operation of the class.

See the *Sapera LT Basic Modules Reference Manual* for a description of all parameters and their possible values.

SapPixPro | SapPixPro Class Members | Basic Class Hierarchy Chart

SapPixProParams

The SapPixProParams Class acts as a container for storing pixel processor parameters used by the SapPixPro Class. These parameters allow you to choose a function (or configuration) in a standard design file.

#include <SapClassBasic.h>

SapPixProParams Class Members | Basic Class Hierarchy Chart

SapPixProParams Class Members

Construction	
SapPixProParams	Class constructor
Attributes	
GetFunctionIndex, SetFunctionIndex	Gets/sets the current function index in a standard design file
GetConfigIndex, SetConfigIndex	Gets/sets the current configuration index in a standard design file
GetBankId, SetBankId	Gets/sets the memory bank identifier used in the Pixel Processor
GetRefImage, SetRefImage	Gets/sets the reference image acquisition mode into the pixel processor memory bank
GetWeightingFactor, SetWeightingFactor	Gets/sets the weighting factor used by the 'Weighting Average' function
GetIntegrateCount, SetIntegrateCount	Gets/sets the integration count used by the 'Integrate and Scale' function
GetIntegrateFactor, SetIntegrateFactor	Gets/sets the integration factor used by the 'Integrate and Scale' function

SapPixProParams | Basic Class Hierarchy Chart

Member Functions

The following functions are members of the SapPixProParams Class.

SapPixProParams::SapPixProParams

SapPixProParams(
);

Remarks

The SapPixProParams constructor initializes its members to default (but probably incorrect) values. Use the other methods in this class to properly set these values.

SapPixProParams | SapPixProParams Class Members | Basic Class Hierarchy Chart

SapPixProParams::GetFunctionIndex, SapPixProParams::SetFunctionIndex

int **GetFunctionIndex**();

void **SetFunctionIndex**(int *index*);

Remarks

Gets/sets the current function index in a standard design file. If this index is set to –1, the SapPixPro Class uses the configuration index instead.

SapPixProParams | SapPixProParams Class Members | Basic Class Hierarchy Chart

SapPixProParams::GetConfigIndex, SapPixProParams::SetConfigIndex

int **GetConfigIndex**();

void **GetConfigIndex**(int *index*);

Remarks

Gets/sets the current configuration index in a standard design file. The SapPixPro Class only uses this index if the function index is set to –1.

SapPixProParams | SapPixProParams Class Members | Basic Class Hierarchy Chart

SapPixProParams::GetBankId, SapPixProParams::SetBankId

int **GetBankId**();

void **SetBankId**(int *bankId*);

Remarks

Gets/sets the memory bank identifier used in the Pixel Processor.

SapPixProParams | SapPixProParams Class Members | Basic Class Hierarchy Chart

SapPixProParams::GetRefImage, SapPixProParams::SetRefImage

BOOL **GetRefImage**();

void **SetRefImage**(BOOL *isRefImage*);

Remarks

Gets/sets the reference image acquisition mode into the pixel processor memory bank

SapPixProParams | SapPixProParams Class Members | Basic Class Hierarchy Chart

SapPixProParams::GetWeightingFactor, SapPixProParams::SetWeightingFactor

int **GetWeightingFactor**();

void **SetWeightingFactor**(int *factor*);

Remarks

Gets/sets the weighting factor used by the ‘Weighting Average’ function

SapPixProParams | SapPixProParams Class Members | Basic Class Hierarchy Chart

SapPixProParams::GetIntegrateCount, SapPixProParams::SetIntegrateCount

int **GetIntegrateCount**();

void **SetIntegrateCount**(int *count*);

Remarks

Gets/sets the integration count used by the ‘Integrate and Scale’ function

SapPixProParams | SapPixProParams Class Members | Basic Class Hierarchy Chart

SapPixProParams::GetIntegrateFactor, SapPixProParams::SetIntegrateFactor

int **GetIntegrateFactor**();

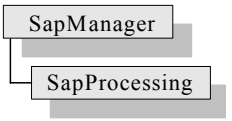
void **GetIntegrateFactor**(int *factor*);

Remarks

Gets/sets the integration factor used by the 'Integrate and Scale' function

SapPixProParams | SapPixProParams Class Members | Basic Class Hierarchy Chart

SapProcessing



SapProcessing Class is the base class required to implement your own processing. This class cannot be used directly. Rather, derive your own processing class (for example, SapMyProcessing), override the Run method, and insert your custom processing code You should then call the Execute method from inside your SapTransfer callback method.

The SapProcessing Class is a ‘real-time processing template’ that simplifies the synchronization between the transfer task and the processing task.

When the Run method is called, you may easily retrieve the index of the next buffer resource that is ready to process. You then simply have to put your custom processing code in the overridden SapProcessing::Run method.

An internal processing thread optimizes buffer processing in real-time. This allows the main application thread to execute without any concerns for the processing task.

An ‘auto empty’mechanism allows synchronization between SapProcessing and SapTransfer objects in order to process buffers in real-time without missing any data.

#include <SapClassBasic.h>

SapProcessing Class Members | Base Class: SapManager | Basic Class Hierarchy Chart

SapProcessing Class Members

Construction	
SapProcessing	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Attributes	
GetBuffer, SetBuffer	Gets/sets the SapBuffer object with the buffer resources to process
SetCallbackInfo	Sets the application callback method to call after processing each buffer, and the associated context
GetCallback	Gets the current application callback method
GetContext	Gets the application context associated with the application callback method
GetTime	Gets the execution time for the most recently processed buffer
GetIndex	Gets the index of the current or last processed buffer
IsAutoEmpty, SetAutoEmpty	Gets/sets the 'auto-empty' mechanism
GetThreadPriority, SetThreadPriority	Gets/sets the execution priority of the processing thread
Operations	
Init	Initializes the processing index
Execute	Process the next buffer or a specific one, possibly skipping buffers in the process
ExecuteNext	Process the next buffer, without skipping any buffers in the process
Run	Method overridden in application code to implement custom processing

SapProcessing | Base Class: SapManager | Basic Class Hierarchy Chart

Member Functions

The following functions are members of the SapProcessing Class.

SapProcessing::SapProcessing

SapProcessing(
  SapBuffer **pBuffer*,
  SapProCallback *pCallback = NULL*,
  void **pContext = NULL*
);

Parameters

pBuffer	SapBuffer object with the buffer resources to process
pCallback	Application callback function to be called after each buffer has been processed. The callback function must be declared as: void MyCallback(SapProCallbackInfo **pInfo*);
pContext	Optional pointer to an application context to be passed to the callback function. If *pCallback* is NULL, this parameter is ignored.

Remarks

The SapProcessing constructor does not actually create the low-level Sapera resources. To do this, you must call the Create method.

This class cannot be instantiated directly. You must first derive a new class from it (for example, SapMyProcessing), override the Run method, and then put your custom processing code within that method.

SapProcessing | SapProcessing Class Members | Basic Class Hierarchy Chart

SapProcessing::Create

virtual BOOL **Create**();

Return Value

Returns TRUE if the object was successfully created, FALSE otherwise

Remarks

Creates all the low-level Sapera resources needed by the processing object. Also initializes the processing buffer index using the current SapBuffer index. Always call this method after SapBuffer::Create.

SapProcessing | SapProcessing Class Members | Basic Class Hierarchy Chart

SapProcessing::Destroy

virtual BOOL **Destroy**();

Return Value

Returns TRUE if the object was successfully destroyed, FALSE otherwise

Remarks

Destroys all the low-level Sapera resources needed by the processing object. Always call this method before SapBuffer::Destroy.

SapProcessing | SapProcessing Class Members | Basic Class Hierarchy Chart

SapProcessing::GetBuffer, SapProcessing::SetBuffer

SapBuffer ***GetBuffer**();

virtual BOOL **SetBuffer**(SapBuffer **pBuffer*);

Parameters

pBuffer SapBuffer object containing the buffer resources to process

Remarks

Gets/sets the SapBuffer object with the buffer resources to process. You can only call SetBuffer before the Create method.

SapProcessing | SapProcessing Class Members | Basic Class Hierarchy Chart

SapProcessing::SetCallbackInfo

virtual BOOL **SetCallbackInfo**(SapProCallback *pCallback*, void **pContext* = NULL);

Remarks

Sets the application callback method to call after processing each buffer, and the associated context.

You can only call SetCallbackInfo before the Create method. See the SapProcessing constructor for more details.

SapProcessing | SapProcessing Class Members | Basic Class Hierarchy Chart

SapProcessing::GetCallback

SapProCallback **GetCallback**();

Remarks

Gets the current application callback method.

The initial value for this attribute is NULL, unless you specify another value in the constructor.

See the SapProcessing constructor for more details.

SapProcessing | SapProcessing Class Members | Basic Class Hierarchy Chart

SapProcessing::GetContext

void ***GetContext**();

Remarks

Gets the application context associated with the application callback method.

The initial value for this attribute is NULL, unless you specify another value in the constructor.

See the SapProcessing constructor for more details.

SapProcessing | SapProcessing Class Members | Basic Class Hierarchy Chart

SapProcessing::GetTime

float **GetTime**();

Remarks

Gets the execution time for the most recently processed buffer (in milliseconds).

The initial value for this attribute is 0.

SapProcessing | SapProcessing Class Members | Basic Class Hierarchy Chart

SapProcessing::GetIndex

int **GetIndex**();

Remarks

When you call GetIndex from within the Run method of your custom processing class, it returns the index of the current buffer to process. When you call it at any other time, it returns the index of the last processed buffer.

SapProcessing | SapProcessing Class Members | Basic Class Hierarchy Chart

SapProcessing::IsAutoEmpty, SapProcessing::SetAutoEmpty

virtual BOOL **IsAutoEmpty**();

virtual void **SetAutoEmpty**(BOOL *isAutoEmpty*);

Remarks

Gets/sets the 'auto-empty' mechanism, used for synchronizing the transfer and processing tasks in the application program.

By default, the SapTransfer Class automatically calls SapBuffer::SetState (SapBuffer::StateEmpty) after an image has been acquired into a buffer. This means that a new image could be acquired in the same buffer before the processing task can even process it.

In order to correctly synchronize the transfer and processing tasks, you must first disable this behavior by calling SapTransfer::IsAutoEmpty, SapTransfer::SetAutoEmpty SapTransfer::SetAutoEmpty(FALSE). Then call SapProcessing::SetAutoEmpty(TRUE) to enable it in this class instead.

As a result, no images will be acquired in the current buffer as long as the Run method is executing. The buffer state is then reset before the application callback method, if any, is called.

The initial value for this attribute is FALSE, unless you construct this object using an existing SapProcessing object.

SapProcessing | SapProcessing Class Members | Basic Class Hierarchy Chart

SapProcessing::GetThreadPriority, SapProcessing::SetThreadPriority

int **GetThreadPriority**();

virtual void **SetThreadPriority**(int *priority*);

Remarks

Gets/sets the execution priority of the processing thread.

The initial value for this attribute is normal priority, unless you construct this object using an existing SapProcessing object.

For a detailed description of this setting, refer to the SetThreadPriority function in the Win32 documentation.

SapProcessing | SapProcessing Class Members | Basic Class Hierarchy Chart

SapProcessing::Init

virtual void **Init**();

Remarks

Initializes the processing index from the current buffer index. The Create method automatically performs this action. This ensures correct synchronization between the processing and buffer index. So you normally do not have to call Init.

However, if you use the ExecuteNext method, but do not call it for every frame, then the processing index will not be synchronized with the buffer index. In such a case you must call Init explicitly to restore synchronization.

SapProcessing | SapProcessing Class Members | Basic Class Hierarchy Chart

SapProcessing::Execute

virtual void **Execute**();

virtual void **Execute**(int *index*);

Parameters

index Index of the buffer resource to process

Remarks

If the *index* is specified, the corresponding buffer in the SapBuffer object is processed through the internal processing thread and the Run method. Otherwise, the current buffer is processed.

If you want to process data acquired in real-time in a buffer through the SapTransfer Class, simply call the Execute method within the SapTransfer callback function in the application code. This will eventually call the Run method in your derived processing class.

The SapProcessing Class will then process as many frames as possible without slowing down the transfer process. This means that some buffers will be skipped if the processing task is too slow to keep up with the acquisition. If you need all frames to be processed, call the ExecuteNext method instead.

SapProcessing | SapProcessing Class Members | Basic Class Hierarchy Chart

SapProcessing::ExecuteNext

virtual void **ExecuteNext**();

Remarks

This method processes the next unprocessed buffer in the SapBuffer object through the internal processing thread and the Run method.

If you want to process data acquired in real-time into a buffer through the SapTransfer Class, simply call the ExecuteNext method within the SapTransfer callback method. This will eventually call the Run method in your derived processing class.

SapProcessing Class will then process all the frames and possibly slow down the transfer process if needed. If the processing task is fast enough to keep-up with the incoming frames, ExecuteNext behaves exactly the same way as Execute. Otherwise, the transfer process must be slowed down to give the SapProcessing object the chance to process every frame.

If you want to process as many frames as possible without affecting the transfer process, use the Execute method instead.

SapProcessing | SapProcessing Class Members | Basic Class Hierarchy Chart

SapProcessing::Run

virtual BOOL **Run**() = 0;

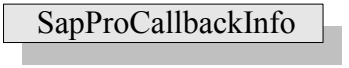
Remarks

This method is automatically invoked by the internal processing thread whenever a buffer is available for processing.

You first need to derive your own class from SapProcessing. Then override Run, and add your own processing code to it. You can use the GetIndex method to get the index of the buffer to process.

SapProcessing | SapProcessing Class Members | Basic Class Hierarchy Chart

SapProCallbackInfo



The SapProCallbackInfo Class acts as a container for storing all arguments to the callback function for the SapProcessing Class.

#include <SapClassBasic.h>

SapProCallbackInfo Class Members | Basic Class Hierarchy Chart

SapProCallbackInfo Class Members

Construction	
SapProCallbackInfo	Class constructor
Attributes	
GetProcessing	Gets the SapProcessing object associated with the processing callback function
GetContext	Gets the application context associated with the SapProcessing callback function

SapProCallbackInfo | Basic Class Hierarchy Chart

Member Functions

The following functions are members of the SapProCallbackInfo Class.

SapProCallbackInfo::SapProCallbackInfo

SapProCallbackInfo(
  SapProcessing **pPro*,
  void **context*
);

Parameters

pPro	SapProcessing object that calls the callback function
context	Pointer to the application context

Remarks

SapProcessing objects create an instance of this class before each call to the application callback method, in order to combine all function arguments into one container.

SapProcessing uses this class when notifying the application that a buffer has been fully processed.

SapProCallbackInfo | SapProCallbackInfo Class Members | Basic Class Hierarchy Chart

SapProCallbackInfo::GetProcessing

SapProcessing ***GetProcessing**();

Remarks

Gets the SapProcessing object associated with the processing callback function. See the SapProcessing constructor for more details.

SapProCallbackInfo | SapProCallbackInfo Class Members | Basic Class Hierarchy Chart

SapProCallbackInfo::GetContext

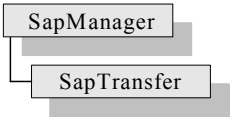
void ***GetContext();**

Remarks

Gets the context information associated with the application callback function. See the SapProcessing constructor for more details.

SapProCallbackInfo | SapProCallbackInfo Class Members | Basic Class Hierarchy Chart

SapTransfer



The SapTransfer Class implements functionality for managing a generic transfer process, that is, the action of transferring data from one source node to a destination node.

All the following classes derived from the SapXferNode Class are considered to be transfer nodes: SapAcquisition, SapBuffer, SapBufferRemote, SapBufferRoi, SapBufferWithTrash, SapCab, SapDsp, and SapPixPro

There are also a number of Specialized Transfer Classes available, for example, SapAcqToBuf. These classes are all derived from SapTransfer, and they may be used to implement common transfer configurations.

#include <SapClassBasic.h>

SapTransfer Class Members | Base Class: SapManager | Basic Class Hierarchy Chart

SapTransfer Class Members

Construction	
SapTransfer	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Attributes	
GetLocation	Returns the location where the transfer resource is located
AddPair	Adds a new pair of source and destination transfer nodes
GetNumPairs	Gets the number of pairs of source and destination transfer nodes
GetPair	Gets access to a specific transfer pair
RemoveAllPairs	Removes all transfer pairs
SetCallbackInfo	Sets the application callback method for transfer events and the associated context
SetTrashCallbackInfo	Sets the trash buffer application callback method for transfer events
GetCallback	Gets the current application callback function for transfer events

GetTrashCallback	Gets the current trash buffer application callback function for transfer events
GetContext	Gets the application context associated with transfer events
IsGrabbing	Checks whether continuous data transfer is currently in progress
IsAutoEmpty, SetAutoEmpty	Gets/sets the 'auto-empty' mechanism
IsAutoConnect SetAutoConnect	Gets/sets automatic activation of physical transfer data paths in the Create method
GetStartMode SetStartMode	Gets/set the synchronization mode used when starting a data transfer
GetCounterStampInfo	Gets the destination buffer counter stamp capabilities for a specific transfer pair
GetHandle	Gets the low-level Sapera handle of the transfer resource
Operations	
Init	Performs the setup for data transfers
Connect	Activates the physical transfer data paths
Disconnect	Deactivates the physical transfer data paths
Select	Sets the current source and destination resource indexes
Snap	Transfers a predetermined number of frames
Grab	Starts continuous data transfer
Freeze	Issues a stop request for continuous data transfer
Abort	Stops the data transfer immediately using brute force
Wait	Waits for complete termination of data transfer
IsCapabilityValid	Checks for the availability of a low-level Sapera C library capability
IsParameterValid	Checks for the availability of a low-level Sapera C library parameter
GetCapability	Gets the value of a low-level Sapera C library capability
GetParameter SetParameter	Gets/sets the value of a low-level Sapera C library parameter

SapTransfer | Base Class: SapManager | Basic Class Hierarchy Chart

Member Functions

The following functions are members of the SapTransfer Class.

SapTransfer::SapTransfer

SapTransfer(
  SapXferCallback *pCallback* = NULL,
  void **pContext* = NULL,
  SapLocation *loc* = SapLocation::ServerUnknown
);

SapTransfer(
  SapXferCallback *pCallback*,
  SapXferCallback *pTrashCallback*,
  void **pContext*,
  SapLocation *loc* = SapLocation::ServerUnknown
);

Parameters

pCallback	Application callback function to be called each time a transfer event happens. The callback function must be declared as: void MyCallback(SapXferCallbackInfo **pInfo*);
pTrashCallback	Application callback function to be called each time a trash buffer transfer event happens
pContext	Optional pointer to an application context to be passed to the callback function. If *pCallback* is NULL, this parameter is ignored.
loc	SapLocation object specifying the server on which the transfer resource is to be created

Remarks

The SapTransfer constructor does not actually create the low-level Sapera resources. To do this, you must call the Create method.

You can use the Specialized Transfer Classes instead of using this class directly, since they simplify the process of instantiating SapTransfer objects that correspond to common transfer configurations.

If you use this class , you must use the AddPair method to add transfer pairs of source and destination nodes. You must do this before calling the Create method.

Trash buffer functionality is only available when a SapBufferWithTrash object is used as a destination transfer node. In this case, the regular callback function is also used for trash buffers, unless you override it using *pTrashCallback*. If you do not use SapBufferWithTrash , then trash buffer settings are ignored.

The specified *pCallback* and *pContext* apply to all transfer pairs by default, unless you override it for specific

pairs using the SapXferPair::SetCallbackInfo method.

By default, regular and trash buffer callback functions are called at each end of frame event, that is, when a complete image has been transferred. You may specify different event types for regular buffers by calling the SapXferPair::GetEventType, SapXferPair::SetEventType method. You cannot change the event type for trash buffers, however.

The server index of the *loc* argument may be set to SapLocation::ServerUnknown. In this case, the most appropriate server for the low-level transfer resource is automatically selected when you call the Create method. The *loc* resource index is ignored.

SapTransfer | SapTransfer Class Members | Basic Class Hierarchy Chart

SapTransfer::Create

virtual BOOL **Create**();

Return Value

Returns TRUE if the object was successfully created, FALSE otherwise

Remarks

Creates all the low-level Sapera resources needed by the transfer object. Always call this method after the Create methods of source and destination nodes for all transfer pairs.

By default, Create automatically calls the Connect method to activate the physical transfer data paths. Calling SetAutoConnect(FALSE) allows you to change values of transfer parameters (or attributes) through methods in the SapXferPair Class, or through calls to the SetParameter method, after calling Create. You must then call Connect explicitly to complete the setup of the transfer resource.

SapTransfer | SapTransfer Class Members | Basic Class Hierarchy Chart

SapTransfer::Destroy

virtual BOOL **Destroy**();

Return Value

Returns TRUE if the object was successfully destroyed, FALSE otherwise

Remarks

Destroys all the low-level Sapera resources needed by the transfer object. Always call this method before the Destroy methods of source and destination nodes for all transfer pairs.

Note that Destroy automatically calls the Disconnect method to deactivate the physical transfer data paths associated with the transfer object.

SapTransfer | SapTransfer Class Members | Basic Class Hierarchy Chart

SapTransfer::GetLocation

SapLocation **GetLocation**();

Remarks

Returns the location where the transfer resource is located.

If you specify a value for this attribute in the SapTransfer constructor, then it is used as the location of all SapXferPair objects that belong to this transfer object.

If you do not specify a value for this attribute, then it defaults to SapLocation::ServerUnknown. When the Create method is called, each SapXferPair object will then use the most appropriate location using the source and destination transfer nodes for the pair.

SapTransfer | SapTransfer Class Members | Basic Class Hierarchy Chart

SapTransfer::AddPair

BOOL **AddPair**(SapXferPair *&pair*);

Parameters

pair Transfer pair of source and destination nodes

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Adds a new pair of source and destination transfer nodes to the current object.

You can only call AddPair before the Create method. However, you do not need to call it if you are using the Specialized Transfer Classes.

See the SapXferPair Class for more details.

SapTransfer | SapTransfer Class Members | Basic Class Hierarchy Chart

SapTransfer::GetNumPairs

int **GetNumPairs**();

Remarks

Gets the number of pairs of source and destination transfer nodes. This value starts at 0 when the transfer object is constructed, increments by 1 at each call to the AddPair method, and is reset to 0 by the RemoveAllPairs method.

SapTransfer | SapTransfer Class Members | Basic Class Hierarchy Chart

SapTransfer::GetPair

SapXferPair ***GetPair**(int *pairIndex*);

Parameters

pairIndex Index of the desired transfer pair

Remarks

Gets access to a specific pair of source and destination transfer nodes. Valid pair indices go from 0 to the value returned by the GetNumPairs method minus 1.

See the SapXferPair Class for more details.

SapTransfer | SapTransfer Class Members | Basic Class Hierarchy Chart

SapTransfer::RemoveAllPairs

BOOL **RemoveAllPairs**();

Remarks

Removes all pairs of source and destination transfer nodes

You can only call RemoveAllPairs before the Create method or after the Destroy method.

SapTransfer | SapTransfer Class Members | Basic Class Hierarchy Chart

SapTransfer::SetCallbackInfo

virtual BOOL **SetCallbackInfo**(SapXferCallback *pCallback*, void **pContext* = NULL);

Remarks

Sets the application callback method for transfer events and the associated context. You can only call SetCallbackInfo before the Create method.

See the SapTransfer constructor for more details.

SapTransfer | SapTransfer Class Members | Basic Class Hierarchy Chart

SapTransfer::SetTrashCallbackInfo

virtual BOOL **SetTrashCallbackInfo**(SapXferCallback *pTrashCallback*);

Remarks

Sets the application callback function for trash buffer transfer events. If you do not call SetTrashCallbackInfo, trash buffers use the same callback function as regular buffers. The associated context information remains the same as for regular buffers.

If you set the value of this attribute to NULL, then the application will receive no trash buffer callbacks.

You can only call SetTrashCallbackInfo before the Create method.

See also the SapTransfer constructor for more details.

SapTransfer | SapTransfer Class Members | Basic Class Hierarchy Chart

SapTransfer::GetCallback

SapXferCallback **GetCallback**();

Remarks

Gets the current application callback function for transfer events. The initial value for this attribute is NULL unless you specify another value in the constructor.

See the SapTransfer constructor for more details.

SapTransfer | SapTransfer Class Members | Basic Class Hierarchy Chart

SapTransfer::GetTrashCallback

SapXferCallback **GetTrashCallback**();

Remarks

Gets the current trash buffer application callback function for transfer events. This function is the same as the one returned using the GetCallback method, unless you explicitly specified a trash buffer callback function in the SapTransfer constructor or using the SetTrashCallbackInfo method.

The initial value for this attribute is NULL unless you specify another value in the constructor.

See the SapTransfer constructor for more details.

SapTransfer | SapTransfer Class Members | Basic Class Hierarchy Chart

SapTransfer::GetContext

void ***GetContext**();

Remarks

Gets the application context associated with transfer events. This context is the same for regular and trash buffer callback functions, even if you explicitly specified a different trash buffer function in the SapTransfer constructor or using the SetTrashCallbackInfo method.

The initial value for this attribute is NULL unless you specify another value in the constructor.

See the SapTransfer constructor for more details.

SapTransfer | SapTransfer Class Members | Basic Class Hierarchy Chart

SapTransfer::IsGrabbing

virtual BOOL **IsGrabbing**();

Remarks

Returns TRUE if continuous data transfer is in progress, FALSE otherwise. Use the Grab method to initiate continuous transfer.

The value of this attribute is only relevant after calling the Create method. Otherwise, it always returns FALSE.

SapTransfer | SapTransfer Class Members | Basic Class Hierarchy Chart

SapTransfer::IsAutoEmpty, SapTransfer::SetAutoEmpty

virtual BOOL **IsAutoEmpty**();

virtual void **SetAutoEmpty**(BOOL *bAutoEmpty*);

Remarks

Gets/sets the 'auto-empty' mechanism, used for synchronizing the transfer with the processing and/or view tasks in the application program.

By default, this class automatically calls SapBuffer::SetState(SapBuffer::StateEmpty) after an image has been acquired into a buffer. This means that a new image could be acquired in the same buffer before the processing or view task can even use it.

In this case, you should call SetAutoEmpty(FALSE) to disable this behavior in this class. You then call SapProcessing:: SetAutoEmpty(TRUE) or SapView::SetAutoEmpty(TRUE), depending on which processing and view task is executed last. Exactly one of the three classes must empty the buffer.

It is also possible to completely disable the auto-empty mechanism for the SapTransfer, SapProcessing, and SapView classes. In this case, you must explicitly call SapBuffer::SetState to empty buffers whenever you have

finished using their contents.

The 'auto-empty' mechanism does not apply when the destination node is not a SapBuffer object.

The initial value for this attribute is TRUE, unless you construct this object using an existing SapTransfer object.

SapTransfer | SapTransfer Class Members | Basic Class Hierarchy Chart

SapTransfer::IsAutoConnect, SapTransfer::SetAutoConnect

virtual BOOL **IsAutoConnect**();

virtual void **SetAutoConnect**(BOOL *bAutoConnect*);

Remarks

Gets/sets automatic activation of physical transfer data paths, so that calling the Create method automatically calls the Connect method when this attribute is TRUE.

Setting auto-connect to FALSE allows you to change values of transfer parameters (attributes) through methods in the SapXferPair Class, or through calls to the SetParameter method, after calling Create. You must then call Connect explicitly to complete the setup of the transfer resource.

The initial value for this attribute is TRUE, unless you construct this object using an existing SapTransfer object.

SapTransfer | SapTransfer Class Members | Basic Class Hierarchy Chart

SapTransfer::GetStartMode, SapTransfer::SetStartMode

virtual SapTransfer::StartMode **GetStartMode**();

virtual BOOL **SetStartMode**(SapTransfer::StartMode *startMode*);

Parameters

startMode The following transfer synchronization modes are available when starting a transfer using the Start or Snap methods:

SapTransfer::StartAsynchronous	Return immediately without waiting for the transfer to begin
SapTransfer::StartSynchronous	For single frame transfers, first wait for any active transfer to end, and return only when the current transfer has been completed.
SapTransfer::StartHalfAsynchronous	For single frame transfers, first wait for any active transfer to end, then immediately return without waiting for the current transfer to begin.

Remarks

Gets/sets the synchronization mode used when starting a data transfer.

The default value for this attribute is StartAsynchronous.

You can only call SetStartMode before the Create method.

SapTransfer | SapTransfer Class Members | Basic Class Hierarchy Chart

SapTransfer::GetCounterStampInfo

const SapXferCounterStampInfo ***GetCounterStampInfo**(int *pairIndex*);

Parameters

pairIndex Index of the desired transfer pair

Remarks

Gets the destination buffer counter stamp capabilities for a specific transfer pair.

The returned SapXferCounterStampInfo object has the following attributes:

BOOL IsSupported()	Returns TRUE if the current transfer device can report these capabilities
BOOL IsAvailable()	Returns TRUE if counter stamp is available
int GetMaxValue()	Returns the maximum counter stamp value
SapXferPair::EventType GetEventType()	Returns the event type that identifies the reference point for the counter stamp. See the SapXferPair::GetEventType, SapXferPair::SetEventType method for a list of possible values.
SapXferPair::CounterStampTimeBase GetTimeBase()	Returns the base units used for the counter stamp. See the SapXferPair::GetCounterStampTimeBase, SapXferPair::SetCounterStampTimeBase method for a list of possible values.

SapTransfer | SapTransfer Class Members | Basic Class Hierarchy Chart

SapTransfer::GetHandle

virtual CORHANDLE **GetHandle**();

Remarks

Gets the low-level Sapera handle of the transfer resource, which you may then use from the low-level Sapera functionality. The handle is only valid after you call the Create method.

See the *Sapera LT Basic Modules Reference Manual* for details on low-level Sapera functionality.

SapTransfer | SapTransfer Class Members | Basic Class Hierarchy Chart

SapTransfer::Init

virtual BOOL **Init**(BOOL *resetIndex* = TRUE);

Parameters

resetIndex TRUE to initialize the buffer index, FALSE otherwise

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Performs the setup for data transfers. Set *resetIndex* to TRUE if you also want to set all destination buffer resources to the empty state, and set the SapBuffer index to the first buffer in its list (through the SapBuffer::ResetIndex method).

You usually do not have to call Init explicitly, since the Create method already does this.

SapTransfer | SapTransfer Class Members | Basic Class Hierarchy Chart

SapTransfer::Connect

virtual BOOL **Connect**();

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Activates the physical transfer data paths associated with a transfer object.

You normally do not need to use this method, as it is called automatically by the Create method. It is useful when used together with the Disconnect method, as in the following case:

```
pXfer->Disconnect();
// Modify some transfer parameters
```

```
pXfer->Connect();
```

This allows the modification of transfer parameters (attributes) through methods in the SapXferPair Class, or through calls to the SetParameter method, since these are not accessible after calling Destroy.

The Create method can also skip the call to Connect altogether, if you first call the SetAutoConnect method to turn off auto-connect, as in the following case:

```
pXfer->SetAutoConnect(FALSE)
;pXfer->Create();
// Modify some transfer parameters
pXfer->Connect();
```


SapTransfer::Disconnect

virtual BOOL **Disconnect**();

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Deactivates the physical transfer data paths associated with a transfer object.

You normally do not need to use Disconnect, as it is called automatically by the Destroy method. It is only useful when used together with the Connect method.

See the Connect method for more details.

SapTransfer::Select

virtual BOOL **Select**(SapXferPair **pPair*, inr *srcIndex* = -1, int *dstIndex* = -1);

virtual BOOL **Select**(int *pairIndex*, inr *srcIndex* = -1, int *dstIndex* = -1);

Parameters

pPair	Pointer to new transfer pair
srcIndex	New resource index for source transfer node
dstIndex	New resource index for destination transfer node
pairIndex	Index of new transfer pair

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Sets a new transfer pair and the current source/destination transfer node resource indexes.

There is usually only one transfer pair per SapTransfer object, in which case the *pairIndex* argument is 0. The source node is usually a SapAcquisition object, in which case the *srcIndex* argument is 0. Since the destination node is usually a SapBuffer object, the *dstIndex* argument then represents a buffer resource index.

Setting *srcIndex* and *dstIndex* to -1 allows for the selection of a new transfer pair while keeping its current source and destination resources indexes.

The Select method is useful in two cases. It allows the selection of pair and resource indexes before changing values of transfer parameters (or attributes) through methods in the SapXferPair Class, or through calls to the SetParameter method. It also allows precise selection of the current transfer node resource indexes before calling the Snap or Grab methods. It is then possible, for example, to know precisely in which buffer resource the next image will be acquired.

SapTransfer | SapTransfer Class Members | Basic Class Hierarchy Chart

SapTransfer::Snap

virtual BOOL **Snap**(int *count* = 1);

Parameters

count Number of frames to be transferred

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Transfers a finite number of frames (usually 1) from the source node to the destination node of all transfer pairs in the current SapTransfer object.

By default, transfers are started asynchronously. You may need to call the Wait method immediately after Snap to ensure correct synchronization. See the SetStartMode method if you need to use a different synchronization mode for single frame transfers (*count* = 1).

If you call the Select method before Snap, then the transfer will be performed using the new current source and destination resource indexes. Otherwise, the transfer will proceed using the indexes from the end of the previous transfer operation (using Snap or Grab). If there is no previous transfer, then appropriate defaults from the call to the Create method will be used.

There is a special case when both the source and destination nodes are SapBuffer objects.

First, only one transfer pair is used. Also, the data transfer is actually a buffer to buffer copy operation, with format conversion if necessary. Finally, the start mode is ignored.

SapTransfer | SapTransfer Class Members | Basic Class Hierarchy Chart

SapTransfer::Grab

virtual BOOL **Grab**();

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Starts a continuous transfer from the source node to the destination node of all transfer pairs in the current SapTransfer object.

Continuous transfers are always started asynchronously, that is, no explicit checking is performed to verify if a previous transfer is still active. If you want to perform this check, then you first need to call the Wait method.

If you call the Select method before Grab, then the transfer will be performed starting at the new current source and destination resources indexes. Otherwise, the transfer will proceed using the indexes from the end of the previous transfer operation (using Snap or Grab). If there is no previous transfer, then appropriate defaults from the call to the Create method will be used.

SapTransfer | SapTransfer Class Members | Basic Class Hierarchy Chart

SapTransfer::Freeze

virtual BOOL **Freeze**();

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Issues a stop request for the current continuous transfer (started with the Grab method). The actual data transfer will end only after the current frame is completely transferred, so you should call the Wait method immediately after Freeze to ensure correct synchronization.

SapTransfer | SapTransfer Class Members | Basic Class Hierarchy Chart

SapTransfer::Abort

virtual BOOL **Abort**();

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Stops data transfers immediately using brute force, without waiting for the current frame to be completely transferred.

You should call Abort only for emergencies. For example, calling Wait after the Snap or Grab methods may fail because of a timeout condition (usually hardware-related). In this case, using Abort is often the only way to correct the situation.

SapTransfer | SapTransfer Class Members | Basic Class Hierarchy Chart

SapTransfer::Wait

virtual BOOL **Wait**(int *timeout*);

Return Value

Returns TRUE if successful, FALSE otherwise

Parameters

timeout Maximum amount of time to wait, in milliseconds

Remarks

Waits for the complete termination of data transfer. You may want to call Wait after Snap to make certain that the required number of frames have been transferred before proceeding. You should definitely call Wait after initiating continuous transfer with Grab and ending it with Freeze.

If the specified *timeout* expires, and transfer is still not completed, then Wait returns an error. A common reason for this error is some kind of hardware failure. In this case, call the Abort method to unconditionally terminate the transfer.

You may also get an error if the *timeout* is too small, and does not give the transfer enough time to terminate gracefully. So you should always specify a value large enough to allow one full frame to be transferred. You may even specify a much larger value (like a few seconds), if your application allows it.

SapTransfer | SapTransfer Class Members | Basic Class Hierarchy Chart

SapTransfer::IsCapabilityValid

virtual BOOL **IsCapabilityValid**(int *cap*);

Parameters

cap Low-level Sapera C library capability to check

Return Value

Returns TRUE if the capability is supported, FALSE otherwise

Remarks

Checks for the availability of a low-level Sapera C library capability for the transfer module. Call this method before GetCapability to avoid invalid or not available capability errors.

IsCapabilityValid is rarely needed. The SapAcquisitionTransfer class already uses important capabilities internally for self-configuration and validation.

See the *Sapera LT Basic Modules Reference Manual* for a description of all capabilities and their possible values.

SapTransfer | SapTransfer Class Members | Basic Class Hierarchy Chart

SapTransfer::IsParameterValid

virtual BOOL **IsParameterValid**(int *param*);

Parameters

param Low-level Sapera C library parameter to check

Return Value

Returns TRUE if the parameter is supported, FALSE otherwise

Remarks

Checks for the availability of a low-level Sapera C library parameter for the transfer module. Call this method before GetParameter to avoid invalid or not available parameter errors.

IsParameterValid is rarely needed. The SapAcquisitionTransfer class already uses important parameters internally for self-configuration and validation.

See the *Sapera LT Basic Modules Reference Manual* for a description of all parameters and their possible values.

SapTransfer | SapTransfer Class Members | Basic Class Hierarchy Chart

SapTransfer::GetCapability

virtual BOOL **GetCapability**(int *cap*, void **pValue*);

Parameters

param	Low-level Sapera C library capability to read
pValue	Pointer to capability value to read back

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

This method allows direct read access to low-level Sapera C library capabilities for the transfer module. It needs a pointer to a memory area large enough to receive the capability value, which is usually a 32-bit integer.

You will rarely need to use GetCapability. The SapTransfer Class already uses important capabilities internally for self-configuration and validation.

See the *Sapera LT Basic Modules Reference Manual* for a description of all capabilities and their possible values.

SapTransfer | SapTransfer Class Members | Basic Class Hierarchy Chart

SapTransfer::GetParameter, SapTransfer::SetParameter

virtual BOOL **GetParameter**(int *param*, void **pValue*);

virtual BOOL **SetParameter**(int *param*, int *value*);

virtual BOOL **SetParameter**(int *param*, void **pValue*);

Parameters

param	Low-level Sapera C library parameter to read or write
pValue	Pointer to parameter value to read back or to write
value	New parameter value to write

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

These methods allow direct read/write access to low-level Sapera C library parameters for the transfer module. The GetParameter method needs a pointer to a memory area large enough to receive the parameter value, which is usually a 32-bit integer. The first form of SetParameter accepts a 32-bit value for the new value. The second form takes a pointer to the new value, and is required when the parameter uses more than 32-bits of storage.

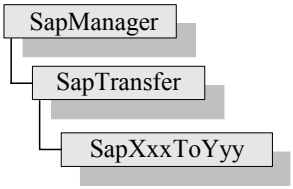
Note that you will rarely need to use these methods. You should first make certain that what you need is not already supported through the SapTransfer or SapXferPair Class. Also, directly setting parameter values may interfere with the correct operation of the class.

Since many parameters cannot be changed when the physical transfer data paths are activated, you may need to use the Disconnect and Connect methods when modifying parameter values. See the Connect method for more details.

See the *Sapera LT Basic Modules Reference Manual* for a description of all parameters and their possible values.

SapTransfer | SapTransfer Class Members | Basic Class Hierarchy Chart

Specialized Transfer Classes



The Specialized Transfer Classes are a set of classes derived from SapTransfer that allow you to more easily create the most commonly used transfer configurations.

All the classes have the same naming convention, that is, SapXxxToYyy, where Xxx and Yyy identify the source and destination nodes, respectively. For example, use the SapAcqToBuf Class to connect a SapAcquisition object to a SapBuffer object.

Each of these classes has one or more specific constructors; otherwise, they use the same methods as SapTransfer.

If you need a transfer configuration that is not supported by any of the specialized classes, then you must use SapTransfer directly instead.

#include <SapClassBasic.h>

Base Class: SapManager | Basic Class Hierarchy Chart

Common Constructor Arguments

All specialized transfer classes constructors include the following two arguments:

pCallback — Application callback function to be called each time a transfer event happens. The callback function must be declared as:

void MyCallback(SapXferCallbackInfo **pInfo*);

pContext — Optional pointer to an application context to be passed to the callback function. If *pCallback* is NULL, this parameter is ignored.

SapAcqToBuf Class

SapAcqToBuf(SapAcquisition *_pAcq_, SapBuffer *_pBuf_, SapXferCallback _pCallback_ = NULL, void *_pContext_ = NULL);

Parameters

pAcq	Source acquisition object
pBuf	Destination buffer object

Remarks

Implements a transfer from an acquisition object to a buffer object

Specialized Transfer Classes | Basic Class Hierarchy Chart

SapAcqDeviceToBuf Class

SapAcqDeviceToBuf(SapAcqDevice *_pAcqDevice_, SapBuffer *_pBuf_, SapXferCallback _pCallback_ = NULL, void *_pContext_ = NULL);

Parameters

pAcqDevice	Source acquisition device object
pBuf	Destination buffer object

Remarks

Implements a transfer from an acquisition device object to a buffer object

Specialized Transfer Classes | Basic Class Hierarchy Chart

SapCabToBuf Class

SapCabToBuf(SapCab *_pCab_, SapBuffer *_pBuf,_ SapXferCallback _pCallback_ = NULL, void *_pContext_ = NULL);

SapCabToBuf(SapCab *_pCab_, int _srcPort_, SapBuffer *_pBuf,_ SapXferCallback _pCallback_ = NULL, void *_pContext_ = NULL);

Parameters

pCab	Source CAB object
srcPort	Data port number on source CAB object
pBuf	Destination buffer object

Remarks

Implements a transfer from a CAB device to a buffer object.

Specialized Transfer Classes | Basic Class Hierarchy Chart

SapBufToBuf Class

SapBufToBuf(SapBuffer *_pSrcBuf_, SapBuffer *_pDstBuf,_ SapXferCallback _pCallback_ = NULL, void *_pContext_ = NULL);

Parameters

pSrcBuf	Source buffer object
pDstBuf	Destination buffer object

Remarks

Implements a transfer from a buffer object to another buffer object. This class is useful when transferring data between the host and an acquisition hardware-based buffer (or an embedded processor). It cannot be used with two buffer objects on the host.

Specialized Transfer Classes | Basic Class Hierarchy Chart

SapAcqToCab Class

SapAcqToCab(SapAcquisition *_pAcq_, SapCab *_pCab_, SapXferCallback _pCallback_ = NULL, void *_pContext_ = NULL);

SapAcqToCab(SapAcquisition *_pAcq_, SapCab *_pCab_, int _dstPort_, SapXferCallback _pCallback_ = NULL, void *_pContext_ = NULL);

Parameters

pAcq	Source acquisition object
pCab	Destination CAB object
dstPort	Data port number on destination CAB object

Remarks

Implements a transfer from an acquisition device to a CAB device.

Specialized Transfer Classes | Basic Class Hierarchy Chart

SapCabToCab Class

SapCabToCab(SapCab **pSrcCab*, SapCab **pDstCab*, SapXferCallback *pCallback* = NULL, void **pContext* = NULL);

Parameters

pSrcCab	Source CAB object
pDstCab	Destination CAB object

Remarks

Implements a transfer from a CAB device to another CAB device.

Specialized Transfer Classes | Basic Class Hierarchy Chart

SapBufToCab Class

SapBufToCab(SapBuffer **pBuf*, SapCab **pCab*, SapXferCallback *pCallback* = NULL, void **pContext* = NULL);

Parameters

pBuf	Source buffer object
pCab	Destination CAB object

Remarks

Implements a transfer from a buffer object to a CAB device.

Specialized Transfer Classes | Basic Class Hierarchy Chart

SapAcqToBufThruDsp Class

SapAcqToBufThruDsp(SapAcquisition *_pAcq_, SapDsp *_pDsp_, SapBuffer *_pBuf_, SapXferCallback _pCallback_ = NULL, void *_pContext_ = NULL);

Parameters

pAcq	Source acquisition object
pDsp	Intermediate DSP object
pBuf	Destination buffer object

Remarks

Implements a transfer from an acquisition device through a DSP device to a buffer object.

Specialized Transfer Classes | Basic Class Hierarchy Chart

SapAcqToBufThruPixPro Class

SapAcqToBufThruPixPro(SapAcquisition *_pAcq_, SapPixPro *_pPixPro_, SapBuffer *_pBuf_, SapXferCallback _pCallback_ = NULL, void *_pContext_ = NULL);

Parameters

pAcq	Source acquisition object
pPixPro	Intermediate pixel processor object
pBuf	Destination buffer object

Remarks

Implements a transfer from an acquisition device through a pixel processor device to a buffer object.

Specialized Transfer Classes | Basic Class Hierarchy Chart

SapAcqToCabThruDsp Class

SapAcqToCabThruDsp(SapAcquisition *_pAcq_, SapDsp *_pDsp_, SapCab *_pCab_, SapXferCallback _pCallback_ = NULL, void *_pContext_ = NULL);

Parameters

pAcq	Source acquisition object
pDsp	Intermediate DSP object
pCab	Destination CAB object

Remarks

Implements a transfer from an acquisition device through a DSP device to a CAB device.

Specialized Transfer Classes | Basic Class Hierarchy Chart

SapAcqToCabThruPixPro Class

SapAcqToCabThruPixPro(SapAcquisition *_pAcq_, SapPixPro *_pPixPro_, SapCab *_pCab_,
SapXferCallback _pCallback_ = NULL, void *_pContext_ = NULL);

Parameters

pAcq	Source acquisition object
pPixPro	Intermediate pixel processor object
pCab	Destination CAB object

Remarks

Implements a transfer from an acquisition device through a pixel processor device to a CAB device.

Specialized Transfer Classes | Basic Class Hierarchy Chart

SapAcqToBufCab Class

SapAcqToBufCab(SapAcquisition *_pAcq_, SapBuffer *_pBuf_, SapCab *_pCab_,
SapXferCallback _pCallback_ = NULL, void *_pContext_ = NULL);

SapAcqToBufCab(SapAcquisition *_pAcq_, SapBuffer *_pBuf_, SapCab *_pCab_, int _dstPort_,
SapXferCallback _pCallback_ = NULL, void *_pContext_ = NULL);

Parameters

pAcq	Source acquisition object
pBuf	Destination buffer object
pCab	Destination CAB object
dstPort	Data port number on destination CAB object

Remarks

Implements a transfer from an acquisition device to both a buffer object and a CAB device (in parallel).

Specialized Transfer Classes | Basic Class Hierarchy Chart

SapCabToBufCab Class

SapCabToBufCab(SapCab **pSrcCab*, SapBuffer **pBuf*, SapCab **pDstCab*,
SapXferCallback *pCallback* = NULL, void **pContext* = NULL);

SapCabToBufCab(SapCab **pSrcCab*, int *srcPort*, SapBuffer **pBuf*, SapCab **pDstCab*,
SapXferCallback *pCallback* = NULL, void **pContext* = NULL);

Parameters

pSrcCab	Source CAB object
srcPort	Data port number on source CAB object
pBuf	Destination buffer object
pDstCab	Destination CAB object

Remarks

Implements a transfer from a CAB device to both a buffer object and another CAB device (in parallel).

Specialized Transfer Classes | Basic Class Hierarchy Chart

SapBufToBufCab Class

SapBufToBufCab(SapBuffer **pSrcBuf*, SapBuffer **pDstBuf*, SapCab **pCab*,
SapXferCallback *pCallback* = NULL, void **pContext* = NULL);

Parameters

pSrcBuf	Source buffer object
pDstBuf	Destination buffer object
pCab	Destination CAB object

Remarks

Implements a transfer from a buffer object to both another buffer object and a CAB device (in parallel).

Specialized Transfer Classes | Basic Class Hierarchy Chart

SapAcqToBufCabThruDsp Class

SapAcqToBufCabThruDsp(SapAcquisition *_pAcq_, SapDsp *_pDsp_, SapBuffer *_pBuf_, SapCab *_pCab_, SapXferCallback _pCallback_ = NULL, void *_pContext_ = NULL);

Parameters

pAcq	Source acquisition object
pDsp	Intermediate DSP object
pBuf	Destination buffer object
pCab	Destination CAB object

Remarks

Implements a transfer from an acquisition device through a DSP device to both a buffer object and a CAB device (in parallel).

Specialized Transfer Classes | Basic Class Hierarchy Chart

SapAcqToBufCabThruPixPro Class

SapAcqToBufCabThruDsp(SapAcquisition *_pAcq_, SapPixPro *_pPixPro_, SapBuffer *_pBuf_, SapCab *_pCab_, SapXferCallback _pCallback_ = NULL, void *_pContext_ = NULL);

Parameters

pAcq	Source acquisition object
pPixPro	Intermediate pixel processor object
pBuf	Destination buffer object
pCab	Destination CAB object

Remarks

Implements a transfer from an acquisition device through a pixel processor device to both a buffer object and a CAB device (in parallel).

Specialized Transfer Classes | Basic Class Hierarchy Chart

SapMultiAcqToBuf Class

SapMultiAcqToBuf(SapAcquisition *_pAcq_[], SapBuffer *_pBuf_[], int _numPairs_,
SapXferCallback _pCallback_ = NULL, void *_pContext_ = NULL);

Parameters

pAcq	List of source acquisition objects
pBuf	List of destination buffer object
numPairs	Number of entries in acquisition and buffer lists

Remarks

Implements a transfer from a series of acquisition devices to a matching number of buffer objects. There is a one-to-one relationship between items in the source list and items in the destination list.

All acquisition devices must be located on the same server, that is, comparing their SapLocation attributes using the SapManager::IsSameServer method return TRUE.

Specialized Transfer Classes | Basic Class Hierarchy Chart

SapMultiAcqToCab Class

SapMultiAcqToCab(SapAcquisition *_pAcq_[], SapCab *_pCab_, int _numPairs_,
SapXferCallback _pCallback_ = NULL, void *_pContext_ = NULL);

SapMultiAcqToCab(SapAcquisition *_pAcq_[], SapCab *_pCab_, int _dstPort_[], int _numPairs_,
SapXferCallback _pCallback_ = NULL, void *_pContext_ = NULL);

Parameters

pAcq	List of source acquisition objects
pCab	Destination CAB object
dstPort	List of data port numbers on destination CAB object
numPairs	Number of entries in acquisition and port lists

Remarks

Implements a transfer from a series of acquisition devices to a matching number of destination ports on a CAB device. There is a one-to-one relationship between items in the source list and items in the destination list.

If no destination ports are specified, values in the range from (0, _numPairs_ – 1) are automatically used.

All acquisition devices must be located on the same server, that is, comparing their SapLocation attributes using the SapManager::IsSameServer method return TRUE.

Specialized Transfer Classes | Basic Class Hierarchy Chart

SapMultiCabToBuf Class

SapMultiCabToBuf(SapCab **pCab*, SapBuffer **pBuf*[], int *numPairs*,
SapXferCallback *pCallback* = NULL,void **pContext* = NULL);

SapMultiCabToBuf(SapCab **pCab*, int *srcPort*[], SapBuffer **pBuf*[], int *numPairs*,
SapXferCallback *pCallback* = NULL, void **pContext* = NULL);

Parameters

pCab	Source CAB object
srcPort	List of data port numbers on source CAB object
pBuf	List of destination buffer objects
numPairs	Number of entries in port and buffer lists

Remarks

Implements a transfer from a series of source ports on a CAB device to a matching number of buffer objects. There is a one-to-one relationship between items in the source list and items in the destination list.

If no source ports are specified, values in the range from (0, *numPairs* – 1) are automatically used.

Specialized Transfer Classes | Basic Class Hierarchy Chart

SapMultiAcqToBufCab Class

SapMultiAcqToBufCab(SapAcquisition **pAcq*[], SapBuffer **pBuf*[], SapCab **pCab*, int *numPairs*, SapXferCallback *pCallback* = NULL,void **pContext* = NULL);

SapMultiAcqToBufCab(SapAcquisition **pAcq*[], SapBuffer **pBuf*[], SapCab **pCab*, int *dstPort*[], int *numPairs*, SapXferCallback *pCallback* = NULL,void **pContext* = NULL);

Parameters

pAcq	List of source acquisition objects
pBuf	List of destination buffer objects
pCab	Destination CAB object
dstPort	List of data port numbers on destination CAB object
numPairs	Number of entries in acquisition, buffer, and port lists

Remarks

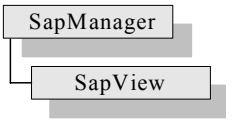
Implements a transfer from a series of acquisition devices to a matching number of buffer objects and CAB devices (in parallel). There is a one-to-one relationship between items in the source list and items in the destination lists.

If no destination ports are specified, values in the range from (0, *numPairs* – 1) are automatically used.

All acquisition devices must be located on the same server, that is, comparing their SapLocation attributes using the SapManager::IsSameServer method returns TRUE.

Specialized Transfer Classes | Basic Class Hierarchy Chart

SapView



The SapView Class includes the functionality to display the resources of a SapBuffer object in a window. It allows you to display the current buffer resource, a specific one, or the next one not yet displayed.

An internal thread optimizes buffer display in realtime. This allows the main application thread to execute without any concerns for the display task.

An 'auto empty' mechanism allows synchronization between SapView and SapTransfer objects to show buffers in realtime without missing any data.

#include <SapClassBasic.h>

SapView Class Members | Base Class: SapManager | Basic Class Hierarchy Chart

SapView Class Members

Construction	
SapView	Class constructor
Create	Allocates the low-level Sapera resources
Destroy	Releases the low-level Sapera resources
Attributes	
GetBuffers SetBuffers	Gets/sets the SapBuffer object with the buffer resources to display
GetDisplay SetDisplay	Gets/sets the SapDisplay object with the display device associated with the view
GetWindow SetWindow	Gets/sets the GDI window handle used for showing buffers
SetCallbackInfo	Sets the application callback method to call after displaying each buffer and the associated context
GetCallback	Gets the current application callback method

GetContext	Gets the application context associated with the application callback method
GetWidth	Gets the width (in pixels) of the displayed buffer area
GetHeight	Gets the height (in lines) of the displayed buffer area
GetViewArea	Gets the width and height of the viewing area
GetScrollPos	Gets the current scrolling position of the viewing area relative to buffer coordinates
GetScrollRange	Gets the scrolling range of the viewing area relative to buffer coordinates
GetIndex	Gets the index of the last displayed buffer
IsAutoEmpty SetAutoEmpty	Gets/sets the 'auto-empty' mechanism
GetOverlayMode SetOverlayMode	Gets/sets the viewing mode when dealing with buffers of overlay type
GetKeyColor SetKeyColor	Gets/sets the keying color when dealing with buffers of overlay type
GetScalingMode SetScalingMode	Gets/sets the mode specifying how buffer content is scaled to the viewing area
GetImmediateMode SetImmediateMode	Gets/sets the view thread bypass mode
GetWindowTitle SetWindowTitle	Gets/sets the title of view windows automatically created by SapView
HasRange	Checks if the view resource can show a subrange of buffer data bits
GetRangeMinMax	Gets the minimum and maximum viewing range values
GetRange SetRange	Gets/sets the viewing range value
GetThreadPriority SetThreadPriority	Gets/sets the execution priority of the viewing thread
GetHandle	Gets the low-level Sapera handle of the view resource
Operations	
Init	Initializes the view index
Show	Shows the next buffer or a specific one, possibly skipping buffers in the process
ShowNext	Shows the next buffer, without skipping any buffers in the process
Hide	Hides the currently displayed buffer
GetDC	Gets the Windows Device Context corresponding to the view window
ReleaseDC	Releases the Windows Device Context corresponding to the view window
GetLut	Gets the current view lookup table

ApplyLut	Programs a new view lookup table
OnPaint	Shows the last displayed buffer again following a WM_PAINT message
OnMove	Adjusts the position of the viewing window following a WM_MOVE message
OnSize	Adjusts the size of the viewing window following a WM_SIZE message
OnHScroll	Adjusts the horizontal scrolling position following a WM_HSCROLL message
OnVScroll	Adjusts the vertical scrolling position following a WM_VSCROLL message
GetCapability	Gets the value of a low-level Sapera C library capability
GetParameter SetParameter	Gets/sets the value of a low-level Sapera C library parameter

SapView | Base Class: SapManager | Basic Class Hierarchy Chart

Member Functions

The following functions are members of the SapView Class.

SapView::SapView

```
SapView(
  SapBuffer *pBuffer = NULL,
  HWND hWnd = SapHwndDesktop,
  SapViewCallback pCallback = NULL,
  void *pContext = NULL
);
SapView(
  SapDisplay *pDisplay,
  SapBuffer *pBuffer,
  HWND hWnd = SapHwndDesktop,
  SapViewCallback pCallback = NULL,
  void *pContext = NULL
);
```

Parameters

pBuffer	SapBuffer object with the buffer resources to display
HWnd	GDI window handle used for displaying buffers
pCallback	Application callback function to be called after each buffer has been displayed. The callback function must be declared as:

void MyCallback(SapViewCallbackInfo *pInfo);

pContext	Optional pointer to an application context to be passed to the callback function. If *pCallback* is NULL, this parameter is ignored.
pDisplay	Displays an object specifying on which display resource the buffers will be shown

Remarks

The SapView constructor does not actually create the low-level Sapera resources. To do this, you must call the Create method.

In addition to a regular window handle, you may use two special values for the *hWnd* argument. If it is equal to SapHwndDesktop, then buffers will be displayed directly on the desktop. If it is equal to SapHwndAutomatic, then SapView will automatically create a view window. The latter is especially useful in console applications, where you do not have a full GUI at your disposal.

You may specify the *pCallback* and *pContext* arguments in order to be notified each time a new buffer is displayed following to a call to the Show, ShowNext, or OnPaint methods. This may be useful if you need to draw graphics in non-destructive overlay.

If you do not specify the *pDisplay* argument, then SapView automatically creates and uses an internal SapDisplay object corresponding to the system display. You must explicitly specify this argument if you use additional SapView objects on the same display, or if you do not want to use the system display.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::Create

virtual BOOL **Create**();

Return Value

Returns TRUE if the object was successfully created, FALSE otherwise

Remarks

Creates all the low-level Sapera resources needed by the view object. Always call this method after SapBuffer::Create.

If you manage the SapDisplay object needed by the view object yourself, you must also call this method after SapDisplay::Create. See the SapView constructor for more details.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::Destroy

virtual BOOL **Destroy**();

Return Value

Returns TRUE if the object was successfully destroyed, FALSE otherwise

Remarks

Destroys all the low-level Sapera resources needed by the view object. Always call this method before SapBuffer::Destroy.

If you manage the SapDisplay object needed by the view object yourself, you must also call this method before SapDisplay::Destroy. See the SapView constructor for more details.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::GetBuffer, SapView::SetBuffer

SapBuffer ***GetBuffer**();

virtual BOOL **SetBuffer**(SapBuffer **pBuffer*);

Parameters

pBuffer SapBuffer object containing the buffer resources to display

Remarks

Gets/sets the SapBuffer object with the buffer resources to display.

You set the initial value for this attribute through the SapView constructor.

You can only call SetBuffer before the Create method.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::GetDisplay, SapView::SetDisplay

SapDisplay ***GetDisplay**();

virtual BOOL **SetDisplay**(SapDisplay **pDisplay*);

Parameters

pDisplay SapDisplay object specifying where the buffer resources are shown

Remarks

Gets/sets the SapDisplay object specifying where the buffer resources are shown.

If you explicitly specify a SapDisplay object in the SapView constructor or through SetDisplay, then GetDisplay returns that object. If you do not, then SapView automatically creates an internal SapDisplay object when calling the Create method, and destroys it when calling the Destroy method. In this case, GetDisplay returns the internal object.

You can only call SetDisplay before the Create method.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::GetWindow, SapView::SetWindow

HWND **GetWindow**();

virtual BOOL **SetWindow**(HWND *hWnd*);

Parameters

hWnd GDI window handle used for displaying buffers

Remarks

Gets/sets the GDI window handle used for displaying buffers.

In addition to a regular window handle, you may use two special values. If *hWnd* is equal to SapHwndDesktop, then buffers will be displayed directly on the desktop. If it is equal to SapHwndAutomatic, then SapView will automatically create a view window. The latter is especially useful in console applications, where you do not have a full GUI at your disposal.

If you do not specify a value for this attribute in the SapView constructor, then it defaults to SapHwndDesktop.

You can only call SetWindow before the Create method.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::SetCallbackInfo

virtual BOOL **SetCallbackInfo**(SapViewCallback *pCallback*, void **pContext* = NULL);

Remarks

Sets the application callback method to call after showing each buffer and the associated context.

You can only call SetCallbackInfo before the Create method. See the SapView constructor for more details.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::GetCallback

SapViewCallback **GetCallback**();

Remarks

Gets the current application callback method called after displaying each buffer. The initial value for this attribute is NULL, unless you specify another value in the constructor.

See the SapView constructor for more details.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::GetContext

void ***GetContext**();

Remarks

Gets the application context associated with the application callback method. The initial value for this attribute is NULL, unless you specify another value in the constructor.

See the SapView constructor for more details.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::GetWidth

virtual int **GetWidth**();

Remarks

Gets the width (in pixels) of the displayed buffer area. This value is equal to the minimum of the buffer width and the viewing area width.

The value returned by GetWidth is only relevant after calling the Create method.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::GetHeight

virtual int **GetHeight**();

Remarks

Gets the height (in lines) of the displayed buffer area. This value is equal to the minimum of the buffer height and the viewing area height width.

The value returned by GetHeight is only relevant after calling the Create method.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::GetViewArea

virtual BOOL **GetViewArea**(int **width*, int **height*);

Remarks

Gets the width and height of the viewing area.

The value returned by GetViewArea is only relevant after calling the Create method.

See also the GetWidth and GetHeight methods.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::GetScrollPos

POINT **GetScrollPos**();

Remarks

Gets the current scrolling position (as a Windows POINT structure) of the viewing area relative to buffer coordinates. The initial value is (0,0) and changes automatically through calls to the OnHScroll and OnVScroll methods. The maximum value depends on the scrolling range (see SapView::GetScrollRange).

Depending on the current view scaling mode, the scrolling position remains fixed at (0,0) if the buffer contents fit entirely within the view area.

The value returned by GetScrollPos is only relevant after calling the Create method.

See the SetScalingMode method for details.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::GetScrollRange

SIZE **GetScrollRange**();

Remarks

Gets the scrolling range (as a Windows SIZE structure) of the viewing area relative to buffer coordinates. This range determines the maximum value of the scrolling position

Depending on the current view scaling mode, the scrolling range is initialized from the number of lines and columns of the view buffer that cannot be shown in the view area. If its value is (0,0), then scrolling is disabled.

The value returned by GetScrollRange is only relevant after calling the Create method.

See the SetScalingMode method for details.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::GetIndex

int **GetIndex**();

Remarks

Gets the index of the last displayed buffer. It is initialized to the current buffer index (usually 0) when you call the Create method. From then on, it is automatically updated following calls to the Show or ShowNext methods.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::IsAutoEmpty, SapView::SetAutoEmpty

virtual BOOL **IsAutoEmpty**();

virtual void **SetAutoEmpty**(BOOL *isAutoEmpty*);

Remarks

Gets/sets the 'auto-empty' mechanism, used for synchronizing the transfer and view tasks in the application program.

By default, the SapTransfer Class automatically calls SapBuffer::SetState(SapBuffer::StateEmpty) after an image has been acquired into a buffer. This means that a new image could be acquired in the same buffer before the view task can even show it. Although this is usually not a critical issue, there are cases in which you need to avoid this.

In order to correctly synchronize the transfer and view tasks, you must first disable this behavior by calling SapTransfer::SetAutoEmpty(FALSE). Then call SapView::SetAutoEmpty(TRUE) to enable it in this class instead.

As a result, no images will be acquired in the current buffer as long as buffer contents have not been shown following calls to the Show or ShowNext methods. The buffer state is then reset before the application callback method, if any, is called.

The initial value for this attribute is FALSE, unless you construct this object using an existing SapView object.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::GetOverlayMode, SapView::SetOverlayMode

SapView::OverlayMode **GetOverlayMode**();

virtual BOOL **SetOverlayMode**(SapView::OverlayMode *overlayMode*);

Parameters

overlayMode Viewing mode for buffers of overlay type, can be one of the following values:

SapView::OverlayNone
Overlay mode is not initialized yet

SapView::OverlayAlwaysOnTop
No color keying scheme is enabled. Buffer contents are displayed directly using the display adapter overlay hardware. This is the fastest method; however, other windows will not be displayed correctly if they overlap the Sapera application.

SapView::OverlayAutoKeying
A destination color keying scheme is enabled. Source buffer pixels are displayed only if the corresponding pixel on the display has the key color. Each time a buffer is shown following calls to the Show or ShowNext methods, the current keying color is painted on the view surface. Also, the OnPaint method only repaints the keying color on the part of the view area that becomes visible again. This is usually the default mode.

SapView::OverlayManualKeying
Similar to auto-keying mode, except that you are responsible for painting the key color in the view area. This gives you more flexibility as to where the overlay image should be displayed.

Remarks

Gets/sets the viewing mode when dealing with buffers of overlay type (SapBuffer::TypeOverlay).

The initial value for this attribute is OverlayNone. If you do not call SetOverlayMode before the Create method,. then the latter will initialize its value appropriately.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::GetKeyColor, SapView::SetKeyColor

SapDataRGB **GetKeyColor**();

virtual BOOL **SetKeyColor**(SapDataRGB *keyColor*);

Remarks

Gets/sets the keying color when dealing with buffers of overlay type (SapBuffer::TypeOverlay). See the SapDataRGB Class for a description of the related data type.

For an 8-bit display mode, that is, when the SapDisplay::GetPixelDepth method returns 8, then only the red color component is relevant.

The initial value for this attribute is black. When calling the Create method, if the current viewing mode is overlay, then its value will be initialized using the current low level keying color value.

You can only call SetKeyColor after the Create method.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::GetScalingMode, SapView::SetScalingMode

SapView::ScalingMode **GetScalingMode**();

virtual BOOL **SetScalingMode**(SapView::ScalingMode *scalingMode*, BOOL *keepAspectRatio* = FALSE);

virtual BOOL **SetScalingMode**(float *zoomHorz*, float *zoomVert*);

virtual BOOL **SetScalingMode**(SapViewScaleParams &*srcParams*, SapViewScaleParams &*dstParams*);

Parameters

scalingMode	SapView:: ScalingNone	There is a one-to-one correspondence between buffer data and pixels shown in the view area. This is the default mode.
	SapView:: ScalingFitToWindow	Displayed buffer contents are scaled so that they are shown completely in the view area. This results in distorted images if the width/height aspect ratio of the buffer is different from the aspect ratio of the view area.
	SapView:: ScalingZoom	Displayed buffer contents are scaled independently in the horizontal and vertical directions
	SapView:: ScalingUserDefined	Buffer contents are displayed using custom user-specified settings
keepAspectRatio	Set to TRUE to keep the image aspect ratio when using ScalingFitToWindow mode	
zoomHorz	Horizontal zooming factor to apply to displayed buffer contents	
zoomVert	Vertical zooming factor to apply to displayed buffer contents	
srcParams	Buffer area to be shown in the specified region of the viewing area	

dstParams Region of the viewing area that will show the specified buffer area

Remarks

Gets/sets the mode specifying how buffer content is scaled to the viewing area.

The first form of SetScalingMode allows you to specify one of two predefined modes: a one-to-one relationship between buffer contents and the view area (ScalingNone), or displaying buffer contents completely (ScalingFitToWindow).

The second form allows you to specify independent horizontal and vertical scaling factors (ScalingZoom). These apply to displayed images only, they do not affect buffer data. This results in distorted images if the factors are different.

The third form gives you complete control over the scaling mode (ScalingUserDefined). You need to specify the exact region in the source buffer, and the exact region in the destination view area. SapView then automatically calculates the appropriate horizontal and vertical scaling factors.

The *srcParams* and *dstParams* arguments both define rectangular areas, as follows:

SapViewScaleParams(int *left*, int *top*, int *width*, int *height*)

The initial value for this attribute is ScalingNone.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::GetImmediateMode, SapView::SetImmediateMode

BOOL **GetImmediateMode**();

virtual void **SetImmediateMode** (BOOL *immediateMode*);

Remarks

Gets/sets the view thread bypass mode.

By default, this mode is off, therefore calling the Show and ShowNext methods wake up an internal thread to handle buffer display. Since showing images is often a time-consuming process, this allows the calling thread to do other things instead.

If immediate mode is active, then the Show and ShowNext methods bypass the thread, and images are shown in the context of the calling thread instead.

The initial value for this attribute is FALSE.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::GetWindowTitle, SapView::SetWindowTitle

virtual BOOL **GetWindowTitle**(char **title*);

virtual void **SetWindowTitle** (const char **title*);

Remarks

Gets/sets the title of view windows automatically created by this class. This is the case when you specify *hWnd* equal to SapHwndAutomatic in the SapView constructor, or if you use the SetWindow method to accomplish the same goal.

When using GetWindowTitle, make certain that the destination string can hold at least 128 characters.

You can only call GetWindowTitle and SetWindowTitle after the Create method.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::HasRange

virtual BOOL **HasRange**();

Remarks

Checks if the view resource can show a subrange of buffer data bits. This is useful when the number of significant bits is less than the number of bit per pixel for the buffer, for example, data coming from a 10-bit camera stored in a 16-bit buffer.

Use the SetRange method to set the viewing range value.

You can only call HasRange after the Create method.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::GetRangeMinMax

virtual void **GetRangeMinMax**(int **pRangeMin*, int **pRangeMax*);

Parameters

pRangeMin	Pointer to returned minimum range value
pRangeMax	Pointer to returned maximum range value

Remarks

Gets the minimum and maximum viewing range values allowed for the SetRange method. If both values are 0, then you cannot change the range.

You can only call GetRangeMinMax after the Create method.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::GetRange, SapView::SetRange

virtual int **GetRange**();

virtual BOOL **SetRange**(int *range*);

Remarks

Gets/sets the viewing range value. Before using GetRange and SetRange, you should first check for availability of this feature using the HasRange and GetRangeMinMax methods.

The range value is the number of bits (starting from the most significant) that are not shown on the display. The default value is 0, that is, the most significant bits are shown. This is a problem when not all bits are used, for example, 10-bit data stored in the low-order bits of a 16-bit buffer. In this case, you should set the value to 6 for correct results.

You can only call GetRange and SetRange after the Create method.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView:: GetThreadPriority, SapView::SetThreadPriority

int **GetThreadPriority**();

virtual void **SetThreadPriority**(int *priority*);

Remarks

Gets/sets the execution priority of the view thread.

The initial value for this attribute is normal priority, unless you construct this object using an existing SapView object.

For a detailed description of this setting, refer to the SetThreadPriority function in the Win32 documentation.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::GetHandle

virtual CORHANDLE **GetHandle**();

Remarks

Gets the low-level Sapera handle of the view resource, which you may then use from the low-level Sapera functionality. The handle is only valid after you call the Create method.

See the *Sapera LT Basic Modules Reference Manual* for details on low-level Sapera functionality.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::Init

virtual void **Init**();

Remarks

Initializes the view index from the current buffer index. The Create method automatically performs this action. This ensures correct synchronization between the view and buffer index. Therefore, you normally do not have to call Init.

However, if you use the ShowNext method, but do not call it for every frame, then the view index will not be synchronized with the buffer index. In such a case you must call Init explicitly to restore synchronization.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::Show

virtual void **Show**();

virtual void **Show**(int *index*);

Parameters

index Index of the buffer resource to show

Remarks

If the *index* is specified, the corresponding buffer in the SapBuffer object is shown through the internal view thread. Otherwise, the current buffer is shown.

If the SapBuffer object has only one buffer resource, that is, if the SapBuffer::GetCount method returns 1, then *index* is ignored, and is assumed to be 0.

If you want to display data acquired in realtime in a buffer through the SapTransfer Class, simply call the Show method within the SapTransfer callback function in application code.

The SapView Class will then show as many frames as possible without slowing down the transfer process. This means that some buffers will be skipped if the view task is too slow to keep up with the acquisition. If you need all frames to be shown, call the ShowNext method instead.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::ShowNext

virtual void **ShowNext**();

Remarks

This method shows the next undisplayed buffer in the SapBuffer object through the internal view thread.

If you want to display data acquired in real-time into a buffer through the SapTransfer Class, simply call the ShowNext method within the SapTransfer callback method.

The SapView Class will then show all the frames and possibly slow down the transfer process if needed. If the view task is fast enough to keep-up with the incoming frames, ShowNext behaves exactly the same way as Show. Otherwise, the transfer process must be slowed down to give the SapView object the chance to show every frame.

If you want to show as many frames as possible without affecting the transfer process, use the Show method instead.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::Hide

virtual void **Hide**();

Remarks

Hides the currently displayed buffer. This is only relevant when dealing with buffers of overlay type (SapBuffer::TypeOverlay).

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::GetDC

virtual BOOL **GetDC**(HDC **pDC*);

Parameters

pDC Pointer to display context value

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Gets the Windows Device Context corresponding to the current view area.

If the current SapView object does not use the system display (see SapDisplay::GetType), then GetDC returns the Windows Device Context corresponding to the entire display instead.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::ReleaseDC

virtual BOOL **ReleaseDC**();

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Releases the Windows Device Context corresponding to the current view area.

If the current SapView object does not use the system display (see SapDisplay::GetType), then ReleaseDC releases the Windows Device Context corresponding to the entire display instead.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::GetLut

SapLut ***GetLut**();

Remarks

Gets the current view lookup table, which has already been automatically created and initialized when calling the Create method. You may manipulate the LUT through the methods in the SapLut Class, and reprogram it using the ApplyLut method.

GetLut returns NULL if the current view resource does not support lookup tables.

This feature is currently available only when the SapDisplay object associated with the view is not located on the primary VGA in the system (see SapDisplay::IsPrimaryVGABoard).

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::ApplyLut

virtual BOOL **ApplyLut**();

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

Reprograms the view lookup table. After getting the current LUT using the GetLut method, use the methods in the SapLut Class to manipulate it. Then use ApplyLut to apply the changes.

This feature is currently available only when the SapDisplay object associated with the view is not located on the primary VGA in the system (see SapDisplay::IsPrimaryVGABoard).

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::OnPaint

virtual void **OnPaint**();

Remarks

Call this method from your application's WM_PAINT message handler to show the last displayed buffer again.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::OnMove

virtual void **OnMove**();

Remarks

Call this method from your application's WM_MOVE message handler to adjusts the position of the viewing window

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::OnSize

virtual void **OnSize**();

Remarks

Call this method from your application's WM_SIZE message handler to adjusts the size of the viewing window

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::OnHScroll

virtual void **OnHScroll**(int *hPosition*);

Parameters

hPosition New horizontal scrolling position

Remarks

Call this method from your application's WM_HSCROLL message handler to adjusts the horizontal scrolling position.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::OnVScroll

virtual void **OnVScroll**(int *vPosition*);

Parameters

vPosition New vertical scrolling position

Remarks

Call this method from your application's WM_VSCROLL message handler to adjusts the vertical scrolling position.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::GetCapability

virtual BOOL **GetCapability**(int *cap*, void **pValue*);

Parameters

param Low-level Sapera C library capability to read

pValue Pointer to capability value to read back

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

This method allows direct read access to low-level Sapera C library capabilities for the View Module. It needs a pointer to a memory area large enough to receive the capability value, which is usually a 32-bit integer.

You will rarely need to use GetCapability. The SapView Class already uses important capabilities internally for self-configuration and validation.

See the *Sapera LT Basic Modules Reference Manual* for a description of all capabilities and their possible values.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapView::GetParameter, SapView::SetParameter

virtual BOOL **GetParameter**(int *param*, void **pValue*);

virtual BOOL **SetParameter**(int *param*, int *value*);

virtual BOOL **SetParameter**(int *param*, void **pValue*);

Parameters

param	Low-level Sapera C library parameter to read or write
pValue	Pointer to parameter value to read back or to write
value	New parameter value to write

Return Value

Returns TRUE if successful, FALSE otherwise

Remarks

These methods allow direct read/write access to low-level Sapera C library parameters for the View Module. The GetParameter method needs a pointer to a memory area large enough to receive the parameter value, which is usually a 32-bit integer. The first form of SetParameter accepts a 32-bit value for the new value. The second form takes a pointer to the new value, and is required when the parameter uses more than 32-bits of storage.

Note that you will rarely need to use these methods. You should first make certain that what you need is not already supported through the SapView Class. Also, directly setting parameter values may interfere with the correct operation of the class.

See the *Sapera LT Basic Modules Reference Manual* for a description of all parameters and their possible values.

SapView | SapView Class Members | Basic Class Hierarchy Chart

SapViewCallbackInfo

The SapViewCallbackInfo Class acts as a container for storing all arguments to the callback function for the SapView Class.

#include <SapClassBasic.h>

SapViewCallbackInfo Class Members | Basic Class Hierarchy Chart

SapViewCallbackInfo Class Members

Construction	
SapViewCallbackInfo	Class constructor
Attributes	
GetView	Gets the SapView object associated with the view callback function
GetContext	Gets the application context associated with the SapView callback function

SapViewCallbackInfo | Basic Class Hierarchy Chart

Member Functions

The following functions are members of the SapViewCallbackInfo Class.

SapViewCallbackInfo::SapViewCallbackInfo

```
SapViewCallbackInfo(
  SapView *pView,
  void *context
);
```

Parameters

pView	SapView object that calls the callback function
context	Pointer to the application context

Remarks

SapView objects create an instance of this class before each call to the application callback method, in order to combine all function arguments into one container.

SapView uses this class when notifying the application that a buffer has been shown.

SapViewCallbackInfo | SapViewCallbackInfo Class Members | Basic Class Hierarchy Chart

SapViewCallbackInfo::GetView

```
SapView *GetView();
```

Remarks

Gets the SapView object associated with the view callback function. See the SapView constructor for more details.

SapViewCallbackInfo | SapViewCallbackInfo Class Members | Basic Class Hierarchy Chart

SapViewCallbackInfo::GetContext

```
void *GetContext();
```

Remarks

Gets the context information associated with the application callback function. See the SapView constructor for more details.

SapViewCallbackInfo | SapViewCallbackInfo Class Members | Basic Class Hierarchy Chart

SapXferCallbackInfo

The SapXferCallbackInfo Class acts as a container for storing all arguments to the callback function for the SapTransfer Class.

#include <SapClassBasic.h>

SapXferCallbackInfo Class Members | Basic Class Hierarchy Chart

SapXferCallbackInfo Class Members

Construction	
SapXferCallbackInfo	Class constructor
Attributes	
GetTransfer	Gets the SapTransfer object associated with transfer events
GetContext	Gets the application context associated with transfer events
GetEventType	Gets the transfer events that triggered the call to the application callback
GetEventCount	Gets the current count of transfer events
IsTrash	Checks if the current transfer event is associated with a trash buffer
GetPairIndex	Gets the index of the transfer pair associated with the current transfer event

SapXferCallbackInfo | Basic Class Hierarchy Chart

Member Functions

The following functions are members of the SapXferCallbackInfo Class.

SapXferCallbackInfo::SapXferCallbackInfo

```
SapXferCallbackInfo(
  SapTransfer *pXfer,
  void *context,
  SapTransfer::EventType eventType,
  int eventCount,
  BOOL isTrash,
  int pairIndex
);
```

Parameters

pXfer	SapTransfer object that calls the callback function
context	Pointer to the application context
eventType	Combination of transfer events. See the SapXferPair::GetEventType method for a list of possible values.
eventCount	Current transfer event count
isTrash	TRUE if the current transfer event is associated with a trash buffer, FALSE otherwise
pairIndex	Transfer pair index for the current transfer event

Remarks

SapTransfer objects create an instance of this class before each call to the transfer callback method in order to combine all function arguments into one container.

The *pContext* argument takes the value specified in the SapTransfer Class constructor; *eventType* identifies the combination of events that triggered the call to the callback function; and *eventCount* increments by one at each call, starting at 1. The counter is reinitialized each time you call the SapTransfer::Snap or SapTransfer::Grab method.

By default, the event count is associated with the destination node for the transfer. This usually corresponds to a buffer object, and each buffer resource in the object gets its own count. The SapXferPair::GetEventCountSource, SapXferPair::SetEventCountSource method allows the count to be associated with the source node instead. Since this usually corresponds to an acquisition object, the count then increases at every acquired frame.

The *pairIndex* argument identifies the transfer pair associated with the callback. *isTrash* is only relevant when the destination node for this pair is a SapBufferWithTrash object.

SapXferCallbackInfo | SapXferCallbackInfo Class Members | Basic Class Hierarchy Chart

SapXferCallbackInfo::GetTransfer

SapTransfer ***GetTransfer**();

Remarks

Gets the SapTransfer object associated with transfer events. See the SapTransfer constructor for more details.

SapXferCallbackInfo | SapXferCallbackInfo Class Members | Basic Class Hierarchy Chart

SapXferCallbackInfo::GetContext

void ***GetContext**();

Remarks

Gets the application context associated with transfer events. See the SapTransfer constructor for more details.

SapXferCallbackInfo | SapXferCallbackInfo Class Members | Basic Class Hierarchy Chart

SapXferCallbackInfo::GetEventType

SapXferPair::EventType **GetEventType**();

Remarks

Gets the combination of transfer events that triggered the call to the application callback. See the SapXferPair::GetEventType method for the list of possible values.

SapXferCallbackInfo | SapXferCallbackInfo Class Members | Basic Class Hierarchy Chart

SapXferCallbackInfo::GetEventCount

int **GetEventCount**();

Remarks

Gets the current count of transfer events. The initial value is 1 and increments after every call to the transfer callback function. The counter is reinitialized each time you call the SapTransfer::Snap or SapTransfer::Grab method.

By default, the event count is associated with the destination node for the transfer. This usually corresponds to a buffer object, and each buffer resource in the object gets its own count. The SapXferPair::GetEventCountSource, SapXferPair::SetEventCountSource method allows the count to be associated with the source node instead. Since this usually corresponds to an acquisition object, the count then increases at every acquired frame.

SapXferCallbackInfo | SapXferCallbackInfo Class Members | Basic Class Hierarchy Chart

SapXferCallbackInfo::IsTrash

BOOL **IsTrash**();

Remarks

Checks if the current transfer event is associated with a trash buffer. This is only relevant when the destination node for the current pair is a SapBufferWithTrash object.

SapXferCallbackInfo | SapXferCallbackInfo Class Members | Basic Class Hierarchy Chart

SapXferCallbackInfo::GetPairIndex

int **GetPairIndex**();

Remarks

Gets the index of the transfer pair associated with the current transfer event

SapXferCallbackInfo | SapXferCallbackInfo Class Members | Basic Class Hierarchy Chart

SapXferPair

The SapXferPair Class describes a pair of source and destination transfer nodes.

If your application uses the SapTransfer Class directly, then you must add transfer pairs yourself before calling the SapTransfer::Create method. If your application uses one of the Specialized Transfer Classes instead, then the class constructor adds all the pairs automatically.

#include <SapClassBasic.h>

SapXferPair Class Members | Basic Class Hierarchy Chart

SapXferPair Class Members

Construction	
SapXferPair	Class constructor
Attributes	
GetSrc	Gets the source node for this pair
GetSrcPort	Gets the source node port number for this pair (CAB only)
GetSrcIndex	Gets the source node resource index for this pair
GetDst	Gets the destination node for this pair
GetDstPort	Gets the destination node port number for this pair (CAB only)
IsRegCallback	Checks if a callback method will be registered for this transfer pair
SetCallbackInfo	Sets the application callback method for transfer events and the associated context
GetCallback	Gets the current application callback function for transfer events
GetContext	Gets the application context associated with transfer events
GetEventType SetEventType	Gets/sets the combination of registered transfer event types
GetEventCountSource	Gets/sets the resource type at which the count of transfer events increases

SetEventCountSource	
GetCycleMode SetCycleMode	Gets/sets the buffer cycling mode when the destination node is a SapBuffer object
GetFlipMode SetFlipMode	Gets/sets the flipping (that is, mirroring) mode for transferred images
GetCounterStampTimeBase SetCounterStampTimeBase	Gets/sets the base units used for counter stamps of destination buffers
GetFramesPerCallback SetFramesPerCallback	Gets/sets the number of transferred frames that trigger a call to the application callback
GetFramesOnBoard SetFramesOnBoard	Gets/sets the number of internal buffers to be used between the source and the destination node
GetFlatFieldCycleMode SetFlatFieldCycleMode	Gets/sets the mode used by the acquisition device for using multiple sets of flat-field correction coefficients

SapXferPair | Basic Class Hierarchy Chart

Member Functions

The following functions are members of the SapXferPair Class.

SapXferPair::SapXferPair

```
SapXferPair(
  SapXferNode *pSrc,
  SapXferNode *pDst,
  BOOL regCallback = TRUE
);
SapXferPair(
  SapXferNode *pSrc,
  int srcPort,
  SapXferNode *pDst,
  int dstPort,
  BOOL regCallback = TRUE
);
```

Parameters

pSrc	Source node for this pair
pDst	Destination node for this pair
regCallback	If TRUE, a callback method will be registered for this pair

srcPort	Source node port number or resource index for this pair
dstPort	Destination node port number for this pair

Remarks

The SapXferPair constructor defines a transfer pair as a combination of one source and one destination node, both of which are objects derived from the SapXferNode Class. This means they can be objects of one of the following classes: SapAcquisition, SapBayer, SapBuffer, SapBufferRemote, SapBufferRoi, SapBufferWithTrash, SapCab, SapDsp, SapPixPro.

If *regCallback* is TRUE, then the SapTransfer object containing this pair automatically registers a callback function when you call the SapTransfer::Create method. By default, the callback function and application context specified in the SapTransfer::SapTransfer constructor are used. You may override these for a specific pair by calling the SetCallbackInfo method in this class.

If *regCallback* is FALSE, then no callback function is registered. Use this option when you do not need notification of transfer events for this pair.

The *srcPort* argument applies to two cases only. If the source node is a SapCab or SapPixPro object, then it identifies the source data port number. If the source node is a SapBuffer object, then it identifies the source buffer resource index. In all other cases, *srcPort* is ignored.

The *dstPort* argument applies only in one case. If the destination node is a SapCab or SapPixPro object, then it identifies the destination data port number. In all other cases, *dstPort* is ignored.

SapXferPair | SapXferPair Class Members | Basic Class Hierarchy Chart

SapXferPair::GetSrc

SapXferNode ***GetSrc**();

Remarks

Gets the source node for this pair as an object derived from the SapXferPair Class. See the SapXferPair constructor for a list of derived classes.

SapXferPair | SapXferPair Class Members | Basic Class Hierarchy Chart

SapXferPair::GetSrcPort

int **GetSrcPort**();

Remarks

Gets the source node port number for this pair. This applies only when the source node is a SapCab or SapPixPro object.

SapXferPair | SapXferPair Class Members | Basic Class Hierarchy Chart

SapXferPair::GetSrcIndex

int **GetSrcIndex**();

Remarks

Gets the source node resource index for this pair. This applies only when the node is a SapBuffer object.

SapXferPair | SapXferPair Class Members | Basic Class Hierarchy Chart

SapXferPair::GetDst

SapXferNode ***GetDst**();

Remarks

Gets the destination node for this pair as an object derived from the SapXferPair Class. See the SapXferPair constructor for a list of derived classes.

SapXferPair | SapXferPair Class Members | Basic Class Hierarchy Chart

SapXferPair::GetDstPort

int **GetDstPort**();

Remarks

Gets the destination node port number for this pair. This applies only when the node is a SapCab or SapPixPro object.

SapXferPair | SapXferPair Class Members | Basic Class Hierarchy Chart

SapXferPair::IsRegCallback

BOOL **IsRegCallback**();

Remarks

Returns TRUE if the SapTransfer object containing this pair automatically registers a callback function when you call the SapTransfer::Create method, FALSE otherwise.

The default value for this attribute is TRUE, unless you specify otherwise in the SapXferPair constructor.

SapXferPair | SapXferPair Class Members | Basic Class Hierarchy Chart

SapXferPair::SetCallbackInfo

BOOL **SetCallbackInfo**(SapXferCallback *pCallback*, void **pContext*);

Remarks

Sets the application callback method for transfer events and the associated context for the current pair only. This overrides any callback and context specified in the SapTransfer constructor.

You can only call SetCallbackInfo before the SapTransfer::Create method or before SapTransfer::Connect if you use SapTransfer::SetAutoConnect to turn off the auto-connect mechanism.

See the SapTransfer constructor for more details.

SapXferPair | SapXferPair Class Members | Basic Class Hierarchy Chart

SapXferPair::GetCallback

SapXferCallback **GetCallback**();

Remarks

Gets the current application callback function for transfer events for the current pair. If NULL, then the callback function specified in the associated SapTransfer object applies to the pair.

The initial value for this attribute is NULL.

SapXferPair | SapXferPair Class Members | Basic Class Hierarchy Chart

SapXferPair::GetContext

void ***GetContext**();

Remarks

Gets the application context associated with transfer events for the current pair. If the value returned by the GetCallback method is NULL, then the context specified in the associated SapTransfer object applies to the pair.

The initial value for this attribute is NULL.

SapXferPair | SapXferPair Class Members | Basic Class Hierarchy Chart

SapXferPair::GetEventType, SapXferPair::SetEventType

SapXferPair::EventType **GetEventType**();

BOOL **SetEventType**(SapXferPair::EventType *eventType*);

Parameters

eventType Transfer events for which the application callback function will be called. One or more of the following values may be combined together using bitwise a OR operation:

SapXferPair::EventNone	No events
SapXferPair::EventStartOfField	Start of field (odd or even)
SapXferPair::EventStartOfOdd	Start of odd field
SapXferPair::EventStartOfEven	Start of even field
SapXferPair::EventStartOfFrame	Start of frame
SapXferPair::EventEndOfField	End of field (odd or even)
SapXferPair::EventEndOfOdd	End of odd field
SapXferPair::EventEndOfEven	End of even field
SapXferPair::EventEndOfFrame	End of frame
SapXferPair::EventEndOfLine	After a specific line number *eventType* = SapXferPair::EventEndOfLine | *lineNum*
SapXferPair::EventEndOfNLines	After a specific line number (linescan cameras only) *eventType* = SapXferPair::EventEndOfNLines | *numLines*
SapXferPair::EventEndOfTransfer	End of transfer, that is, after all frames have been transferred following calls to SapTransfer::Snap or SapTransfer::Grab/SapTransfer::Freeze.
SapXferPair::EventLineUnderrun	The number of active pixels per line received from a video source is less than it should be.
SapXferPair::EventFieldUnderrun	The number of active lines per field received from a video source is less than it should be.

Remarks

SapXferPair | SapXferPair Class Members | Basic Class Hierarchy Chart

SapXferPair::GetEventCountSource, SapXferPair::SetEventCountSource

SapXferPair::EventCountSource **GetEventCountSource**();

BOOL **SetEventCountSource**(SapXferPair::EventCountSource *eventCountSource*);

Parameters

eventCountSource	Resource type where the transfer event count increases. Can be one of the following:	
	SapXferPair::EventCountNone	No event count available
	SapXferPair::EventCountDst	Count is linked to the destination node
	SapXferPair::EventCountSrc	Count is linked to the source node

Remarks

Gets/sets the resource type at which the count of transfer events increases. The destination node normally corresponds to a buffer object, so that each buffer resource in the object gets its own count. The source node usually corresponds to an acquisition object, so that the count increases at every acquired frame.

The initial value for this attribute is EventCountSourceDst. Call SetEventCountSource if you want to override this value for the current transfer pair.

Depending on the current transfer device, you may be allowed to call SetEventCountSource at any time. However, you should still call this method before SapTransfer::Create or before SapTransfer::Connect if you use SapTransfer::SetAutoConnect to turn off the auto-connect mechanism.

SapXferPair | SapXferPair Class Members | Basic Class Hierarchy Chart

SapXferPair::GetCycleMode, SapXferPair::SetCycleMode

SapXferPair::CycleMode **GetCycleMode**();

BOOL **SetCycleMode** (SapXferPair::CycleMode *cycleMode*);

Parameters

cycleMode The available buffer cycling modes differ by the way in which they specify which buffer resource gets the next data transfer.

The empty state refers to the case in which buffer data has been completely processed and may be overwritten. It is set by application code as soon as it has finished processing buffer data.

The full state refers to the case in which buffer data has not been processed since its latest data transfer. It is set by the transfer device as soon as a data transfer has completed.

The current buffer is the one in which the latest data transfer occurred.

The next buffer is the one immediately after the current buffer, with wraparound to the first buffer at the end of the list.

The trash buffer is defined as the last buffer in the list for the WithTrash modes only. Its state is

always considered to be empty by the transfer device.

The cycling mode can be one of the following values:

SapXferPair::CycleUnknown	Unknown cycle mode.
SapXferPair::CycleAsynchronous	Always transfer to the next buffer, regardless of its state.
SapXferPair::CycleSynchronous	If next buffer is empty, then transfer to next buffer; otherwise, transfer to current buffer.
SapXferPair::CycleWithTrash	If next buffer is empty, then transfer to the next buffer; otherwise, transfer to the trash buffer. Repeat transferring to the trash buffer as long as the next buffer is full.
SapXferPair::CycleOff	Always transfer to the current buffer.
SapXferPair::CycleNextEmpty	If next buffer is empty, then transfer to next buffer; otherwise, transfer to next empty buffer in the list. If all buffers are full, then transfer to current buffer.
SapXferPair::CycleNextWithTrash	If next buffer is empty, then transfer to next buffer; otherwise, transfer to next empty buffer in the list. If all buffers are full, then transfer to trash buffer. Repeat transferring to the trash buffer as long as there is no empty buffer in the list.

Remarks

Gets/sets the buffer cycling mode when the destination node is a SapBuffer object.

The initial value for this attribute is CycleUnknown. This means that the associated SapTransfer Class uses a CycleWithTrash cycle mode for a SapBufferWithTrash object; otherwise, it uses CycleAsynchronous. Call SetCycleMode if you want to override this value for the current transfer pair.

Depending on the current transfer device, you may be allowed to call SetCycleMode at any time. However, you should still call this method before SapTransfer::Create or before SapTransfer::Connect if you use SapTransfer::SetAutoConnect to turn off the auto-connect mechanism.

SapXferPair | SapXferPair Class Members | Basic Class Hierarchy Chart

SapXferPair::GetFlipMode, SapXferPair::SetFlipMode

SapXferPair::FlipMode **GetFlipMode**();

BOOL **SetFlipMode**(SapXferPair::FlipMode *flipMode*);

Parameters

	SapXferPair::FlipNone	No flipping
	SapXferPair::FlipHorizontal	Transferred images are flipped horizontally
flipMode	SapXferPair::FlipVertical	Transferred images are flipped vertically

Remarks

Gets/sets the flipping (that is, mirroring) mode for transferred images for the current transfer pair.

The initial value for this attribute is FlipNone.

Depending on the current transfer device, you may be allowed to call SetFlipMode at any time. However, you should still call this method before SapTransfer::Create or before SapTransfer::Connect if you use SapTransfer::SetAutoConnect to turn off the auto-connect mechanism.

SapXferPair | SapXferPair Class Members | Basic Class Hierarchy Chart

SapXferPair::GetCounterStampTimeBase, SapXferPair::SetCounterStampTimeBase

SapXferPair::CounterStampTimeBase **GetCounterStampTimeBase**();

BOOL **SetCounterStampTimeBase** (SapXferPair:: CounterStampTimeBase *counterStampTimeBase*);

Parameters

counterStampTimeBase	Counter stamp units. Can be one of the following:	
	SapXferPair::CounterStampMicroSec	Microseconds
	SapXferPair::CounterStampMilliSec	Milliseconds
	SapXferPair::CounterStampLine	Line valid or horizontal sync signal
	SapXferPair::CounterStampLineTrigger	External line trigger of shaft encoder pulse
	SapXferPair::CounterStampFrame	Frame valid or vertical sync signal
	SapXferPair:: CounterStampExtFrameTrigger	External frame trigger signal

Remarks

Gets/sets the base units used for counter stamps of destination buffers. Individual values have no meaning by themselves; however, subtracting counter stamp values for two buffer resources gives the amount of time (or a number of signal occurrences) elapsed between a common reference point for their respective data transfers.

To find out which common reference point is used for the current transfer pair, see the SapTransfer::GetCounterStampInfo method.

The initial value for this attribute is CounterStampMicroSec. Call SetCounterStampTimeBase if you want to override this value for the current pair.

Depending on the current transfer device, you may be allowed to call SetCounterStampTimeBase at any time. However, you should still call this method before SapTransfer::Create or before SapTransfer::Connect if you use SapTransfer::SetAutoConnect to turn off the auto-connect mechanism.

SapXferPair | SapXferPair Class Members | Basic Class Hierarchy Chart

SapXferPair::GetFramesPerCallback, SapXferPair::SetFramesPerCallback

int **GetFramesPerCallback**();

BOOL **SetFramesPerCallback**(int *numFrames*);

Remarks

Gets/sets the number of transferred frames that trigger a call to the application callback.

This is particularly useful when your application receives a high number of callbacks from the SapTransfer Class, for example, if your acquisition device has a high frame rate. This results in significant CPU overhead, which may negatively affect performance.

In this case, call the SetFramesPerCallback method to reduce the number of calls to your application callback.

The default value for this attribute is 1.

You can only call SetFramesPerCallback before the SapTransfer::Create method or before SapTransfer::Connect if you use SapTransfer::SetAutoConnect to turn off the auto-connect mechanism.

SapXferPair | SapXferPair Class Members | Basic Class Hierarchy Chart

SapXferPair::GetFramesOnBoard, SapXferPair::SetFramesOnBoard

int **GetFramesOnBoard**();

BOOL **SetFramesOnBoard**(int *numFrames*);

Remarks

Gets/sets the number of internal buffers to be used between the source and the destination node. You should always respect the following sequence when calling these two methods:

```
SetFramesOnBoard(numFrames);
SapTransfer::Create();
newNumFrames = GetFramesOnBoard();
```

If the value read back is equal to 0, it means that the transfer hardware has no internal buffers, or that this number cannot be changed. If this value is less than the original *numFrames*, it means that there is not enough internal memory for all the buffers.

Depending on the current transfer device, you may be allowed to call SetFramesOnBoard at any time. However, you should still call this method before SapTransfer::Create or before SapTransfer::Connect if you use SapTransfer::SetAutoConnect to turn off the auto-connect mechanism.

SapXferPair | SapXferPair Class Members | Basic Class Hierarchy Chart

SapXferPair::GetFlatFieldCycleMode, SapXferPair::SetFlatFieldCycleMode

SapXferPair::FlatFieldCycleMode **GetFlatFieldCycleMode**();

BOOL **SetFlatFieldCycleMode**(SapXferPair::FlatFieldCycleMode *flatFieldCycleMode*);

Parameters

flatFieldCycleMode	The flat-field cycling mode can be one of the following values:	
	SapXferPair:: FlatFieldCycleOff	Always use the same set of flat-field gain and offset coefficients
	SapXferPair:: FlatFieldCycleAutomatic	Automatically alternate between sets of flat-field gain and offset coefficients

Remarks

Gets/sets the mode used by the acquisition device for using multiple sets of flat-field correction coefficients. This applies only when a SapFlatField object was created with a number of gain and offset buffers larger than 1, and when hardware correction is available. In this case, the index of the current gain/offset buffer pair increments after each frame has been acquired, and wraps around to 0 when it reaches its maximum value.

The initial value for this attribute is FlatFieldCycleOff.

Depending on the current transfer device, you may be allowed to call SetFlatFieldCycleMode at any time. However, you should still call this method before SapTransfer::Create or before SapTransfer::Connect if you use SapTransfer::SetAutoConnect to turn off the auto-connect mechanism.

SapXferPair | SapXferPair Class Members | Basic Class Hierarchy Chart

SapXferParams

The SapXferParams Class stores parameters needed by a transfer task managed by the SapTransfer Class.

When building a destination transfer node object, use the transfer parameters from the source node to ensure transfer compatibility between the two. You may do this either by specifying the source SapXferNode object in the destination node constructor, or by directly specifying the appropriate SapXferParams object.

All the following classes derived from the SapXferNode Class are considered to be transfer nodes: SapAcquisition, SapBayer, SapBuffer, SapBufferRemote, SapBufferRoi, SapBufferWithTrash, SapCab, SapDsp, SapPixPro.

#include <SapClassBasic.h>

SapXferParams Class Members | Basic Class Hierarchy Chart

SapXferParams Class Members

Construction	
SapXferParams	Class constructor
Attributes	
GetFrameType, SetFrameType	Gets/sets the field interlacing type in a frame
GetFieldOrder, SetFieldOrder	Gets/sets the field order for interlaced frames
GetWidth, SetWidth	Gets/sets the width (in pixels) of one frame
GetHeight, SetHeight	Gets/sets the height (in lines) of one frame
GetFormat, SetFormat	Gets/sets the data format of the transferred data

GetPixelDepth, SetPixelDepth	Gets/sets the number of significant bits of the transferred data
GetPixelShift, SetPixelShift	Gets/sets the difference between the pixel depth and the number of bits in the data format (obsolete)
GetParameters, SetParameters	Gets/sets all the parameters in one operation

SapXferParams | Basic Class Hierarchy Chart

Member Functions

The following functions are members of the SapXferParams Class.

SapXferParams::SapXferParams

SapXferParams(
);

Remarks

The SapXferParams constructor initializes its members to default (but probably incorrect) values. Use the other methods in this class to properly set these values.

SapXferParams | SapXferParams Class Members | Basic Class Hierarchy Chart

SapXferParams::GetFrameType, SapXferParams::SetFrameType

SapXferParams::FrameType **GetFrameType**();

void **SetFrameType**(SapXferParams::FrameType *frameType*);

Parameters

frameType Field interlacing can be one of the following values:

SapXferParams::FrameInterlaced	Video fields are interlaced
SapXferParams::FrameProgressive	Video fields are non-interlaced (progressive video)

Remarks

Gets/sets the field interlacing type in a frame.

SapXferParams | SapXferParams Class Members | Basic Class Hierarchy Chart

SapXferParams::GetFieldOrder, SapXferParams::SetFieldOrder

SapXferParams::FieldOrder **GetFieldOrder**();

void **SetFieldOrder**(SapXferParams::FieldOrder *fieldOrder*);

Parameters

fieldOrder	Field order can be one of the following values:	
	SapXferParams::FieldOrderOddEven	The odd field is transferred before the even field
	SapXferParams::FieldOrderEvenOdd	The even field is transferred before the odd field
	SapXferParams::FieldOrderNext	The next field is transferred, whether it is odd or even

Remarks

Gets/sets the field order for interlaced frames. Does not apply for progressive video.

SapXferParams | SapXferParams Class Members | Basic Class Hierarchy Chart

SapXferParams::GetWidth, SapXferParams::SetWidth

int **GetWidth**();

void **SetWidth**(int *width*);

Remarks

Gets/sets the width (in pixels) of one frame

SapXferParams | SapXferParams Class Members | Basic Class Hierarchy Chart

SapXferParams::GetHeight, SapXferParams::SetHeight

int **GetHeight**();

void **SetHeight**(int *height*) ;

Remarks

Gets/sets the height (in lines) of one frame

SapXferParams | SapXferParams Class Members | Basic Class Hierarchy Chart

SapXferParams::GetFormat, SapXferParams::SetFormat

SapFormat **GetFormat**();

void **SetFormat**(SapFormat *format*);

Remarks

Gets/sets the pixel format of the transferred data. See the SapBuffer::SapBuffer constructor for possible values for *format*.

SapXferParams | SapXferParams Class Members | Basic Class Hierarchy Chart

SapXferParams::GetPixelDepth, SapXferParams::GetPixelDepth

int **GetPixelDepth**();

void **SetPixelDepth**(int *pixelDepth*);

Remarks

Gets/sets the number of significant bits of the transferred data. This value is extracted from SapAcquisition objects to determine the number of bits containing actual data. The range of possible values is given by the SapManager::GetPixelDepthMin, SapManager::GetPixelDepthMax methods.

SapXferParams | SapXferParams Class Members | Basic Class Hierarchy Chart

SapXferParams::GetPixelShift, SapXferParams::SetPixelShift

int **GetPixelShift**();

void **SetPixelShift**(int *pixelShift*);

Remarks

Gets/sets the difference between the pixel depth and the number of bits in the data format for image display purposes.

These methods are **obsolete**, since Sapera LT now automatically manages the image display pixel shift using the buffer pixel depth.

SapXferParams | SapXferParams Class Members | Basic Class Hierarchy Chart

SapXferParams::GetParameters, SapXferParams::SetParameters

void **GetParameters**(SapXferParams::FrameType **frameType*, SapXferParams::FieldOrder **fieldOrder*, int **width*, int **height*, int **format*, int **pixelDepth*, int **pixelShift*);

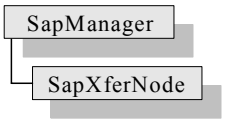
void **SetParameters** (SapXferParams::FrameType *frameType,* SapXferParams::FieldOrder *fieldOrder*, int *width*, int *height*, int *format*, int *pixelDepth*, int *pixelShift*);

Remarks

Gets/sets all the parameters in one operation. See the GetFrameType and GetFieldOrder methods for possible values for *frameType* and *fieldOrder*.

SapXferParams | SapXferParams Class Members | Basic Class Hierarchy Chart

SapXferNode



The SapXferNode Class implements functionality to manipulate a transfer node object. The SapXferPair Class uses two of these objects to create a transfer pair. The SapTransfer Class then uses this pair to implement a transfer configuration.

You should not instantiate SapXferNode objects directly. Rather, you will use one of its derived classes in your applications. All the following classes are derived from SapXferNode: SapAcquisition, SapBayer, SapBuffer, SapBufferRemote, SapBufferRoi, SapBufferWithTrash, SapCab, SapDsp, SapPixPro.

#include <SapClassBasic.h>

SapXferNode Class Members | Base Class: SapManager | Basic Class Hierarchy Chart

SapXferNode Class Members

Construction	
SapXferNode	Class constructor
Attributes	
GetLocation, SetLocation	Gets/sets the location where the transfer node resource is located
GetSrcNode, SetSrcNode	Gets/sets the source transfer node object used for compatibility of parameters with other transfer node objects
GetSrcPort	Gets the source port number for this node
GetXferParams, SetXferParams	Gets/sets the transfer parameters structure used for compatibility of parameters with other transfer node objects
GetHandle	Gets the low-level Sapera handle of the transfer node resource

SapXferNode | Base Class: SapManager | Basic Class Hierarchy Chart

Member Functions

The following functions are members of the SapXferNode Class.

SapXferNode::SapXferNode

SapXferNode(
  SapLocation *loc*
);

SapXferNode(
  SapLocation *loc*,
  SapXferNode **pSrcNode*
);

SapXferNode(
  SapLocation *loc*,
  SapXferParams *xferParams*
);

Parameters

loc	SapLocation object specifying the server where the transfer node resource is located and the index of the resource on this server
pSrcNode	Existing SapXferNode object from which parameters for the current object will be extracted for compatibility of transfer parameters.
xferParams	Transfer parameters structure used for compatibility of parameters with other transfer node objects.

Remarks

You should not instantiate SapXferNode objects directly. Rather, use one of the derived classes in the application. All the following classes are derived from SapXferNode: SapAcquisition, SapBayer, SapBuffer, SapBufferRemote, SapBufferRoi, SapBufferWithTrash, SapCab, SapDsp, and SapPixPro.

SapXferNode | SapXferNode Class Members | Basic Class Hierarchy Chart

SapXferNode::GetLocation, SapXferNode::SetLocation

SapLocation **GetLocation**();

BOOL **SetLocation**(SapLocation *loc*);

Remarks

Gets/sets the location where the transfer node resource is located. You can only call SetLocation before the SapTransfer::Create method.

SapXferNode | SapXferNode Class Members | Basic Class Hierarchy Chart

SapXferNode::GetSrcNode, SapXferNode::SetSrcNode

SapXferNode ***GetSrcNode**();

BOOL **SetSrcNode**(SapXferNode **pSrcNode*, int *srcPort* = 0);

Remarks

Gets/sets the source transfer node object used for compatibility of parameters with other transfer node objects. For SetSrcNode, the optional *srcPort* argument represents the source port number for the node, which applies only to a SapCab or SapPixPro object. You can only call SetSrcNode before the SapTransfer::Create method.

SapXferNode | SapXferNode Class Members | Basic Class Hierarchy Chart

SapXferNode::GetSrcPort

int **GetSrcPort**();

Remarks

Gets the source port number for this node. This applies only to a SapCab or SapPixPro object.

SapXferNode | SapXferNode Class Members | Basic Class Hierarchy Chart

SapXferNode::GetXferParams, SapXferNode::SetXferParams

SapXferParams **GetXferParams**(int *portIndex* = 0);

BOOL **SetXferParams**(SapXferParams *xferParams*, int *portIndex* = 0);

Remarks

Gets/sets the transfer parameters structure used for compatibility of parameters with other transfer node objects. The optional *portIndex* argument represents the port number for the node, which applies only to a SapCab or SapPixPro object. You can only call SapXferParams before the SapTransfer::Create method.

SapXferNode | SapXferNode Class Members | Basic Class Hierarchy Chart

SapXferNode::GetHandle

virtual CORHANDLE **GetHandle**();

Remarks

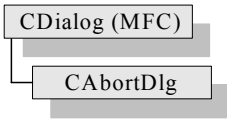
Gets the low-level Sapera handle of the transfer node resource, which you may then use from the low-level Sapera functionality. The exact type of handle depends on the current derived class. The handle is only valid after you call the SapTransfer::Create method.

See the *Sapera LT Basic Modules Reference Manual* for details on low-level Sapera functionality.

SapXferNode | SapXferNode Class Members | Basic Class Hierarchy Chart

GUI Class Reference

CAbortDlg



The CAbortDlg dialog class is a more evolved version of the SapTransfer::Wait method. It allows you to wait an indefinite amount of time, while giving you the possibility to abort at any time. This is useful when the time for one frame is variable (for examplefor example, when using an external trigger).

You may create a modal version of this dialog right after calling SapTransfer::Snap or SapTransfer::Freeze. If the transfer somehow does not terminate cleanly, then clicking the 'Abort' button allows you to call SapTransfer::Abort to unconditionally terminate the transfer.

#include <SapClassGui.h>

CAbortDlg Class Members | GUI Class Hierarchy Chart

CAbortDlg Class Members

Construction

CAbortDlg	Class constructor

CAbortDlg | GUI Class Hierarchy Chart

Member Functions

The following functions are members of the CAbortDlg Class.

CAbortDlg::CAbortDlg

```
CAbortDlg(
  CWnd* pParent,
  SapTransfer *pXfer,
  int timeout = 1000
);
```

Parameters

pParent	Pointer to the parent window of this dialog (CWnd is defined in MFC)
pXfer	Pointer to the related SapTransfer object
timeout	Time to wait before showing the dialog (in milliseconds)

Remarks

The CAbortDlg constructor does not immediately show the dialog. This happens only when you call its DoModal method.

If you call DoModal, and the related transfer is inactive, then the dialog is not shown and the value IDOK is returned right away. The same happens if the transfer ends before the specified *timeout* period.

If you call DoModal, and the related transfer is still active after the *timeout* period, then the dialog appears. If the transfer ends before you click the 'Abort' button, then the dialog automatically closes with a return value equal to IDOK. If you click 'Abort', then the dialog closes with a return value equal to IDCANCEL.

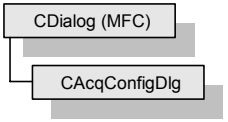
There is usually nothing else to do after the dialog returns IDOK. You, however, must call SapTransfer::Wait if IDCANCEL is returned.

Example

```
m_Xfer->Snap();
if (CAbortDlg(this, m_Xfer).DoModal() != IDOK)
   m_Xfer->Abort();
```

CAbortDlg | CAbortDlg Class Members | GUI Class Hierarchy Chart

CAcqConfigDlg



The CAcqConfigDlg Class allows you to load an acquisition configuration file (CCF) to create a new SapAcquisition object. You may then retrieve this new object using the CAcqConfigDlg:: GetAcquisition method.

Alternatively, you may specify an existing SapAcquisition object. In this case, a copy of the object is automatically created for loading the CCF file. Calling CAcqConfigDlg:: GetAcquisition then returns this object.

#include <SapClassGui.h>

CAcqConfigDlg Class Members | GUI Class Hierarchy Chart

CAcqConfigDlg Class Members

Construction	
CAcqConfigDlg	Class constructor
Attributes	
GetAcquisition	Gets the automatically created SapAcquisition object

CAcqConfigDlg | CAcqConfigDlg Class Members

Member Functions

The following functions are members of the CAcqConfigDlg Class.

CAcqConfigDlg::CAcqConfigDlg

CAcqConfigDlg(
  CWnd* *pParent*,
  SapAcquisition **pAcq* = NULL,
  const char **productDir* = NULL
);

Parameters

pParent	Pointer to the parent window of this dialog (CWnd is defined in MFC)
pAcq	Pointer to the related SapAcquisition object
productDir	Environment variable name that specifies a product directory other than Sapera for CCF files

Remarks

The CAcqConfigDlg constructor does not immediately show the dialog. This happens only when you call its DoModal method.

If you specify an existing SapAcquisition object through the *pAcq* argument, then a new object is automatically created as a working copy. The existing object is not modified.

If *pAcq* is NULL, then the latest acquisition server and resource indices are retrieved from Windows Registry, as well as the name of the CCF file. A new SapAcquisition object is then automatically created from these values. If no such registry entry exists, then the default directory for finding the CCF files is set according to the environment variable you specify with the *productDir* argument (for example, "X64DIR").

When you are finished with the dialog, you can get the new object by calling GetAcquisition.

Example 1: Creating a SapAcquisition object the first time

```
CAcqConfigDlg dlg(this, NULL);
if (dlg.DoModal() == IDOK)
{
   m_Acq = new SapAcquisition(dlg.GetAcquisition());
}
```

Example 2: Modifying an existing SapAcquisition object.

```
CAcqConfigDlg dlg(this, m_Acq);
if (dlg.DoModal() == IDOK)
{
   // Destroys all objects (including m_Acq)
   DestroyObjects();
```

```
    // Creates a backup object
    SapAcquisition acq = *m_Acq;

    // Updates acquisition object
    *m_Acq = dlg.GetAcquisition();

    // Recreates all objects (including m_Acq)
    if (!CreateObjects())
    {
       // If an error occurred, retrieves the backup copy and tries again
       *m_Acq = acq;
       CreateObjects();
    }
}
```

CAcqConfigDlg | CAcqConfigDlg Class Members | GUI Class Hierarchy Chart

CAcqConfigDlg:: GetAcquisition

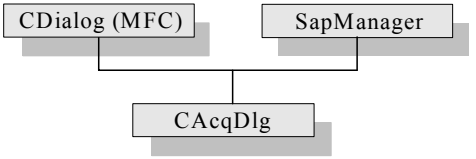
SapAcquisition &**GetAcquisition**();

Remarks

Gets the automatically created SapAcquisition object with the new CCF parameters. This object is not the same as the one specified as a constructor argument, if any.

CAcqConfigDlg | CAcqConfigDlg Class Members | GUI Class Hierarchy Chart

CAcqDlg



The CAcqDlg Class allows you to dynamically adjust the following acquisition parameters:

- Camera selector
- Digital input connect bit ordering
- External trigger options

You may create a modal version of this dialog after calling SapAcquisition::Create.

Changes made through this dialog are saved in the current SapAcquisition object only. You must call the SapAcquisition::SaveParameters method if you need to save the new values to an acquisition configuration file (CCF). Otherwise, reloading a new CCF file (through the CAcqConfigDlg Class) or calling SapAcquisition::Destroy causes all changes to be lost.

#include <SapClassGui.h>

CAcqDlg Class Members | Base Class: SapManager | GUI Class Hierarchy Chart

CAcqDlg Class Members

Construction	
CAcqDlg	Class constructor

CAcqDlg | Base Class: SapManager | GUI Class Hierarchy Chart

Member Functions

The following functions are members of the CAcqDlg Class.

CAcqDlg::CAcqDlg

CAcqDlg(
  CWnd* *pParent*,
  SapAcquisition **pAcq*
);

Parameters

pParent	Pointer to the parent window of this dialog (CWnd is defined in MFC)
pAcq	Pointer to the related SapAcquisition object

Remarks

The CAcqDlg constructor does not immediately show the dialog. This happens only when you call its DoModal method.

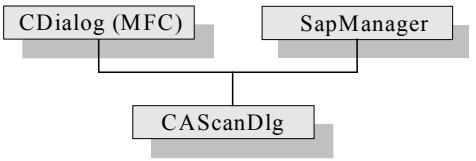
You can only call DoModal after SapAcquisition::Create. Also, you must call SapAcquisition::GetParameter, SapAcquisition::SetParameter after using the dialog if you want to save your changes to an acquisition configuration file (CCF).

Example

```
CAcqDlg dlg(this, m_Acq);
dlg.DoModal();
```

CAcqDlg | CAcqDlg Class Members | GUI Class Hierarchy Chart

CAScanDlg



The CAScanDlg Class allows you to dynamically adjust the following acquisition parameters related to area scan cameras:

- Trigger/reset/integrate options
- Master mode (camera slave) options

You may create a modal version of this dialog after calling SapAcquisition::Create.

Changes made through this dialog are saved in the current SapAcquisition object only. You must call the SapAcquisition::SaveParameters method if you need to save the new values to an acquisition configuration file (CCF). Otherwise, reloading a new CCF file (through the CAcqConfigDlg Class) or calling SapAcquisition::Destroy causes all changes to be lost.

#include <SapClassGui.h>

CAScanDlg Class Members | Base Class: SapManager | GUI Class Hierarchy Chart

CAScanDlg Class Members

Construction

CAScanDlg	Class constructor

CAScanDlg | Base Class: SapManager | GUI Class Hierarchy Chart

Member Functions

The following functions are members of the CAScanDlg Class.

CAScanDlg::CAScanDlg

CAScanDlg(
  CWnd **pParent*,
  SapAcquisition **pAcq*
);

Parameters

pParent	Pointer to the parent window of this dialog (CWnd is defined in MFC)
pAcq	Pointer to the related SapAcquisition object

Remarks

The CAScanDlg constructor does not immediately show the dialog. This happens only when you call its DoModal method.

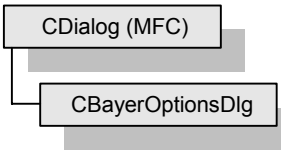
You can only call DoModal after SapAcquisition::Create. Also, you must call SapAcquisition::SaveParameters after using the dialog if you want to save your changes to an acquisition configuration file (CCF).

Example

```
CAScanDlg dlg(this, m_Acq);
dlg.DoModal();
```

CAScanDlg | CAScanDlg Class Members | GUI Class Hierarchy Chart

CBayerOptionsDlg



The CBayerOptionsDlg Class allows you to adjust the following Bayer conversion parameters:

- Alignment mode
- Pixel value calculation method
- Gamma factor for Bayer lookup table
- White balance gain coefficients (manually or automatically)

As opposed to other GUI Classes, the Bayer dialog is modeless, so that you may define a region of interest (for white balance calculation) on the currently displayed image while keeping the dialog visible. You may create this dialog after calling SapBayer::Create.

#include <SapClassGui.h>

CBayerOptionsDlg Class Members | GUI Class Hierarchy Chart

CBayerOptionsDlg Class Members

Construction

CBayerOptionsDlg	Class constructor

CBayerOptionsDlg | GUI Class Hierarchy Chart

Member Functions

The following functions are members of the CBayerOptionsDlg Class.

CBayerOptionsDlg::CBayerOptionsDlg

CBayerOptionsDlg(
  CWnd* *pParent*,

```
  SapBayer *pBayer,
  SapTransfer *pXfer = NULL,
  CImageWnd *pImageWnd = NULL,
  SapProcessing *pPro = NULL
);
```

Parameters

pParent	Pointer to the parent window of this dialog (CWnd is defined in MFC)
pBayer	Pointer to the related SapBayer object (cannot be NULL)
pXfer	Pointer to the related SapTransfer object
pImageWnd	Pointer to the related CImageWnd object
pPro	Pointer to the related SapProcessing object

Remarks

The CBayerOptionsDlg constructor does not immediately show the dialog. This happens only when you call its ShowWindow method.

If *pXfer* is NULL, then a buffer with valid Bayer data must be available through the object pointed to by the *pBayer* argument. This is necessary since a software conversion is performed whenever you perform an automatic white balance calculation, or when you click on the Apply button.

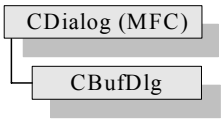
If *pXfer* points to a valid SapTransfer object, then the acquisition device is used to acquire Bayer images.

If *pImageWnd* is NULL, then automatic white balance calculation is not available, as it requires a valid region of interest to be drawn on the image display area.

If you specify a non-NULL *pPro* argument, then the execution time becomes available whenever a software Bayer conversion is performed within the dialog.

CBayerOptionsDlg | CBayerOptionsDlg Class Members | GUI Class Hierarchy Chart

CBufDlg



The CBufDlg Class allows you to modify the parameters of a SapBuffer object. It actually works on a copy of the object, so that modifications do not affect the original object right away.

You may create a modal version of this dialog after calling SapBayer::Create. When you are finished using the dialog, you may retrieve the modified copy of the buffer object using the GetBuffer method.

#include <SapClassGui.h>

CBufDlg Class Members | GUI Class Hierarchy Chart

CBufDlg Class Members

Construction	
CBufDlg	Class constructor
Attributes	
GetBuffer	Gets the modified copy of the original SapBuffer object, as modified from the dialog

CBufDlg | GUI Class Hierarchy Chart

Member Functions

The following functions are members of the CBufDlg Class.

CBufDlg::CBufDlg

CBufDlg(
  CWnd* *pParent*,
  SapBuffer **pBuffer*,
  SapDisplay **pDisplay* = NULL
);

Parameters

pParent	Pointer to the parent window of this dialog (CWnd is defined in MFC)
pBuffer	Pointer to the related SapBuffer object
pDisplay	Pointer to an optional SapDisplay object used for displaying the buffer contents

Remarks

The CBufDlg constructor does not immediately show the dialog. This happens only when you call its DoModal method.

The specified *pBuffer* is not modified when you use the dialog. Instead, an additional SapBuffer object is used internally to keep track of all changes. You may then get the modified object by calling GetBuffer.

If you specify a non-NULL *pDisplay* argument, then this SapDisplay object will be used to automatically determine the availability of offscreen and overlay buffer types for the current buffer format.

You can only call DoModal after SapBuffer::Create.

Example: Modifying an existing SapBuffer object

```
CBufDlg dlg(this, m_Buffers);
if (dlg.DoModal() == IDOK)
{
   // Destroy all objects (including m_Buffers)
   DestroyObjects();

   // Create a backup object
   SapBuffer buf = *m_Buffers;


// Update buffer object
   *m_Buffers = dlg.GetBuffer();

   // Recreate all objects (including m_Buffers)
   if (!CreateObjects())
   {
      // If an error occurred, retrieve the backup copy and try again
      *m_Buffers = buf;
      CreateObjects();
   }
}
```

CBufDlg | CBufDlg Class Members | GUI Class Hierarchy Chart

CBufDlg::GetBuffer

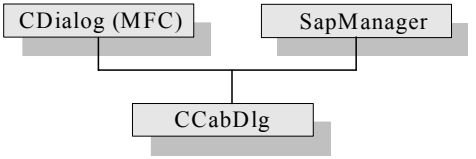
SapBuffer &**GetBuffer**();

Remarks

Gets the modified copy of the original SapBuffer object, as modified from the dialog. This object is not the same as the one used in the CBufDlg constructor, it is a completely different instance.

CBufDlg | CBufDlg Class Members | GUI Class Hierarchy Chart

CCabDlg



The CCabDlg Class allows you to dynamically adjust the following CAB parameters:

- Transmitter/receiver configuration
- Transfer mode
- Block size
- Source/destination channels for each port

You may create a modal version of this dialog after calling SapCab::Create, except if you rely on CCabDlg to automatically create a SapCab object.

Changes made through this dialog are saved in the current SapCab object only. Calling SapCab::Destroy causes all changes to be lost.

#include <SapClassGui.h>

CCabDlg Class Members | Base Class: SapManager | GUI Class Hierarchy Chart

CCabDlg Class Members

Construction	
CCabDlg	Class constructor
Attributes	
GetCab	Gets the original or automatically creates SapCab object

CCabDlg | Base Class: SapManager | GUI Class Hierarchy Chart

Member Functions

The following functions are members of the CCabDlg Class.

CCabDlg::CCabDlg

```
CCabDlg(
  CWnd* pParent,
  SapCab *pCab,
  SapXferNode *pSrcNode = NULL,
  int cabIndex = 0
);
```

Parameters

pParent	Pointer to the parent window of this dialog (CWnd is defined in MFC)
pCab	Pointer to the related SapCab object
pSrcNode	SapXferNode object that provides CAB parameters if *pCab* is NULL
cabIndex	CAB device index to use if *pCab* is NULL

Remarks

The CCabDlg constructor does not immediately show the dialog. This happens only when you call its DoModal method.

You may specify an existing SapCab object through the *pCab* argument. Alternatively, you may set *pCab* to NULL and specify an existing SapXferNode object (usually SapAcquisition) and a CAB device index. In this case, a new SapCab object is automatically created using the same SapLocation attribute as *pSrcNode* and the specified *cabIndex*.

You can only call DoModal after SapCab::Create (except of *pCab* is NULL) and before SapTransfer::Create.

Example

```
m_Xfer->Destroy();

CCabDlg dlg(this, m_Cab);
dlg.DoModal();

m_Xfer->Create();
```

CCabDlg | CCabDlg Class Members | GUI Class Hierarchy Chart

CCabDlg::GetCab

SapCab &**GetCab**();

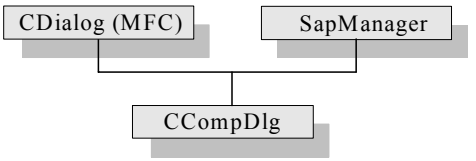
Remarks

If you previously called the CCabDlg constructor with a non-NULL *pCab* argument, this method returns the original SapCab object.

If you called the constructor with *pCab* equal to NULL, but you specified appropriate values for the source node and the CAB device index, then this method returns a new SapCab object automatically created using the same SapLocation attribute as *pSrcNode* and the specified *cabIndex*.

CCabDlg | CCabDlg Class Members | GUI Class Hierarchy Chart

CCompDlg



The CCompDlg Class allows you to dynamically adjust the following acquisition parameters related to composite video input signals:

- Brightness and contrast options
- Hue and saturation options
- Sharpness options

You may create a modal version of this dialog after calling SapAcquisition::Create.

Changes made through this dialog are saved in the current SapAcquisition object only. You must call the SapAcquisition::SaveParameters method if you need to save the new values to an acquisition configuration file (CCF). Otherwise, reloading a new CCF file (through the CAcqConfigDlg Class) or calling SapAcquisition::Destroy causes all changes to be lost.

#include <SapClassGui.h>

CCompDlg Class Members | Base Class: SapManager | GUI Class Hierarchy Chart

CCompDlg Class Members

Construction

CCompDlg	Class constructor

CCompDlg | Base Class: SapManager | GUI Class Hierarchy Chart

Member Functions

The following functions are members of the CCompDlg Class.

CCompDlg::CCompDlg

CCompDlg(
  CWnd **pParent*,
  SapAcquisition **pAcq*,
  SapTransfer **pXfer* = NULL
);

Parameters

pParent	Pointer to the parent window of this dialog (CWnd is defined in MFC)
pAcq	Pointer to the related SapAcquisition object
pXfer	Pointer to a SapTransfer for controlling visual feedback

Remarks

The CCompDlg constructor does not immediately show the dialog. This happens only when you call its DoModal method.

If you supply a non-NULL *pXfer* argument, then a new image is automatically acquired through this SapTransfer object every time you modify one of the values in the dialog. This gives you immediate visual feedback.

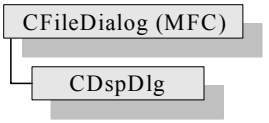
You can only call DoModal after SapAcquisition::Create. Also, you must call SapAcquisition::SaveParameters after using the dialog if you want to save your changes to an acquisition configuration file (CCF).

Example

```
m_Xfer->Snap();
CCompDlg dlg(this, m_Acq, m_Xfer);
dlg.DoModal();
```

CCompDlg | CCompDlg Class Members | GUI Class Hierarchy Chart

CDspDlg



The CDspDlg Class allows you to choose an executable file to load in DSP memory. The Windows Registry is used to remember the name of the most recent DSP file.

You may create a modal version of this dialog after calling SapDsp::Create, except if you rely on CDspDlg to automatically create a SapDsp object.

#include <SapClassGui.h>

CDspDlg Class Members | GUI Class Hierarchy Chart

CDspDlg Class Members

Construction	
CDspDlg	Class constructor
Attributes	
GetDsp	Gets the original or automatically created SapDsp object

CDspDlg | GUI Class Hierarchy Chart

Member Functions

The following functions are members of the CDspDlg Class.

CDspDlg::CDspDlg

CDspDlg(
  CWnd* *pParent*,
  SapDsp **pDsp*,
  SapXferNode **pSrcNode* = NULL,
  int *dspIndex* = 0
);

Parameters

pParent	Pointer to the parent window of this dialog (CWnd is defined in MFC)
pDsp	Pointer to the related SapDsp object
pSrcNode	SapXferNode object that provides DSP parameters if *pDsp* is NULL
dspIndex	DSP device index to use if *pDsp* is NULL

Remarks

The CDspDlg constructor does not immediately show the dialog. This happens only when you call its DoModal method.

You may specify an existing SapDsp object through the *pDsp* argument. Alternatively, you may set *pDsp* to NULL and specify an existing SapXferNode object (usually SapAcquisition) and a DSP device index. In this case, a new SapDsp object is automatically created using the same SapLocation attribute as *pSrcNode*, the specified *dspIndex*, and the name of the most recent DSP file (from the Windows Registry).

You can only call DoModal after SapDsp::Create (except if *pDsp* is NULL) and before SapTransfer::Create.

Example

```
// Both m_Dsp (SapDsp) and m_Acq (SapAcquisition) are used.
// The first time, m_Dsp is NULL and m_Acq is used for initialization.
// The second time, m_Dsp is used for initialization.

CDspDlg dlg(this, m_Dsp, m_Acq);
if (dlg.DoModal() == IDOK)
{
       // Destroy all objects (including m_Dsp)
       DestroyObjects();

   SapDsp dsp;

   if (!m_Dsp)
```

```
    {
       // Allocate new object
       m_Dsp = new SapDsp(dlg.GetDsp());
    }
    else
    {
       // Create a backup object
       dsp = *m_Dsp;

        // Update current object
                *m_Dsp = dlg.GetDsp();
    }

        // Recreate all objects (including m_Dsp)
    if (!CreateObjects())
    {
       // If an error occurred, retrieve the backup copy and try again
       *m_Dsp = dsp;
       if (!CreateObjects())
       {
           delete m_Dsp;
           m_Dsp = NULL;
           CreateObjects();
       }
    }
}
```

CDspDlg | CDspDlg Class Members | GUI Class Hierarchy Chart

CDspDlg::GetDsp

SapDsp &**GetDsp**();

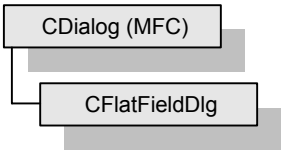
Remarks

If you previously called the CDspDlg constructor with a non-NULL *pDsp* argument, this method returns the original SapDsp object.

If you called the constructor with *pDsp* equal to NULL, but you specified appropriate values for the source node and the DSP device index, then this method returns a new SapDsp object automatically created using the same SapLocation attribute as *pSrcNode*, the specified *dspIndex*, and the name of the most recent DSP file (from the Windows Registry).

CDspDlg | CDspDlg Class Members | GUI Class Hierarchy Chart

CFlatFieldDlg



The CFlatFieldDlg Class allows you to perform flat-field calibration by either acquiring or loading from files black and white reference images.

You may create a modal version of this dialog after calling SapFlatField::Create.

#include <SapClassGui.h>

CFlatFieldDlg Class Members | GUI Class Hierarchy Chart

CFlatFieldDlg Class Members

Construction

CFlatFieldDlg	Class constructor

CFlatFieldDlg | GUI Class Hierarchy Chart

Member Functions

The following functions are members of the CFlatFieldDlg Class.

CFlatFieldDlg::CFlatFieldDlg

CFlatFieldDlg(
  CWnd* *pParent*,
  SapFlatField **pFlatField*,
  SapTransfer **pXfer*,
  SapBuffer **pBuffer*
);

Parameters

pParent	Pointer to the parent window of this dialog (CWnd is defined in MFC)
pFlatField	Pointer to the related SapFlatField object
pXfer	Pointer to the related SapTransfer object
pBuffer	Pointer to the related SapBuffer object

Remarks

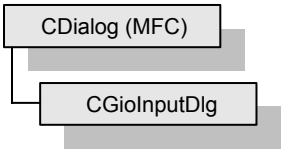
The CFlatFieldDlg constructor does not immediately show the dialog. This happens only when you call its DoModal method.

If *pXfer* is NULL, then the black and the white images needed for computing the flat-field correction coefficients have to be loaded from files.

If *pXfer* points to a valid SapTransfer object, then these images will be acquired from the acquisition device.

CFlatFieldDlg | CFlatFieldDlg Class Members | GUI Class Hierarchy Chart

CGioInputDlg



The CGioInputDlg Class includes the following functionality:

- Gets the current pin state (low or high)
- Gets read/write access to the low-level Sapera C library parameter for the I/O module
- Sets the input level type I/O
- Enables/disables a callback function for specific I/O events (rising edge or falling edge)
- Event count for each occuring I/O event

You may create a modal version of this dialog after calling SapGio::Create except for when you use CGioInputDlg to automatically create a SapGio object.

#include <SapClassGui.h>

CGioInputDlg Class Members | GUI Class Hierarchy Chart

CGioInputDlg Class Members

Construction

CGioInputDlg	Class constructor

GUI Class Hierarchy Chart

Member Functions

The following functions are members of the CGioInputDlg Class.

CGioInputDlg:: CGioInputDlg

CGioInputDlg(
  CWnd* *pParent*,
  UINT *iDevice*,
  SapGio **pGio*
);

Parameters

pParent	Pointer to the parent window of this dialog (CWnd is defined in MFC)
iDevice	Input GIO resource index
pGio	Pointer to the related SapGio object

Remarks

The CGioInputDlg constructor does not immediately show the dialog. This occurs only after you call its DoModal method.

You may specify an existing SapGio object through the pGio argument.

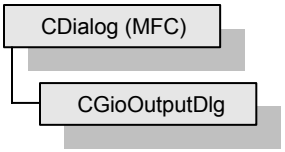
You can only call DoModal after SapGio::Create.

Example: Constructor calls for acquisition devices supporting general I/O

```
CGioInputDlg  *m_pDlgInput[MAX_GIO_DEVICE];
//Loop for all resources
for (UINT32 iDevice = 0; (iDevice < MAX_GIO_DEVICE) && (iDevice < m_gioCount);
iDevice++) {
m_pDlgInput[iDevice] = new CGioInputDlg(this, iDevice, m_pGio[iDevice]);
if (m_pDlgInput[iDevice] != NULL)
              m_pDlgInput[iDevice]->Create();}
```

CGioInputDlg | CGioInputDlg Class Members | GUI Class Hierarchy Chart

CGioOutputDlg



The CGioOutputDlg Class includes the following functionality:

- Gets the number of pins present on the output I/O resource
- Gets/sets the current pin state (low or high)
- Gets/sets the current output type parameter value

You may create a modal version of this dialog after calling SapGio::Create except for when you use CGioOutputDlg to automatically create a SapGio object.

#include <SapClassGui.h>

CGioOutputDlg Class Members | GUI Class Hierarchy Chart

CGioOutputDlg Class Members

Construction

CGioOutputDlg	Class constructor

CGioOutputDlg | GUI Class Hierarchy Chart

Member Functions

The following functions are members of the CGioOutputDlg Class.

CGioOutputDlg::CGioOutputDlg

CGioOutputDlg(
CWnd* *pParent*,
UINT *iDevice*,
SapGio **pGio*);

Parameters

pParent	Pointer to the parent window of this dialog (CWnd is defined in MFC)
iDevice	Output GIO resource index
pGio	Pointer to the related SapGio object

Remarks

The CGioOutputDlg constructor does not immediately show the dialog. This occurs only after you call its DoModal method.

You may specify an existing SapGio object through the pGio argument.

You can only call DoModal after SapGio::Create.

Example: Constructor calls for acquisition devices supporting general I/O

```
CGioOutputDlg *m_pDlgOutput[MAX_GIO_DEVICE];
//Loop for all resources
for (UINT32 iDevice = 0; (iDevice < MAX_GIO_DEVICE) && (iDevice < m_gioCount);
iDevice++) {
m_pDlgOutput[iDevice] = new CGioOutputDlg(this, iDevice, m_pGio[iDevice]);
if (m_pDlgOutput[iDevice] != NULL)
       m_pDlgOutput[iDevice]->Create();}
```

CGioOutputDlg | CGioOutputDlg Class Members | GUI Class Hierarchy Chart

CImageWnd

The CImageWnd utility class includes the following functionality:

- Manipulates an image window (view window)
- Manipulates scroll bars attached to the image window
- Interactively draws a region-of-interest (ROI) in the image window
- Gets a description of the pixel value at the current mouse position in the image window

Since CImageWnd uses a SapView object, you may use it to simplify the processing of application messages related to the view window such as WM_MOVE, WM_PAINT, WM_SIZE, etc.

There are two ways of using CImageWnd:

- The view window and the scroll bars already exist in a dialog-based application. These controls are specified as arguments to the class constructor.
- In non dialog-based application, the view window is the main application window itself. The scroll bars are then accessed through this window.

#include <SapClassGui.h>

CImageWnd Class Members | GUI Class Hierarchy Chart

CImageWnd Class Members

Construction	
CImageWnd	Class constructor
Operations (message handling)	
OnMove	Adjusts the position of the view window following a WM_MOVE message
OnPaint	Repaints uncovered areas of the View window following a WM_PAINT message
OnSize	Adjusts the size of the view window following a WM_SIZE message
OnHScroll	Adjusts the horizontal scroll bar following a WM_HSCROLL message
OnVScroll	Adjusts the vertical scroll bar following a WM_VSCROLL message
OnLButtonDown	Begins or ends drawing of the ROI tracker following a WM_LBUTTONDOWN message
OnSetCursor	Sets the cursor shape according to the position of the mouse over the ROI tracker following a WM_SETCURSOR message
Operations (ROI tracker)	
IsRoiTrackerActive	Checks if the ROI tracker is currently displayed
DisplayRoiTracker	Redraws the ROI tracker
HideRoiTracker	Hides the ROI tracker
GetSelectedRoi	Gets the rectangle corresponding to the ROI tracker
SelectRoi	Sets the ROI tracker rectangle
Operations (other)	
IsPointInside	Checks if the specified point is inside the view window
TranslateMousePos	Converts mouse position coordinates to the view window coordinate system
Invalidate	Invalidates the view window client area
UpdateClientArea	Repaints unused regions of the view window client area
GetPixelString	Gets a text description of the pixel value at a specific position in the view window

CImageWnd | GUI Class Hierarchy Chart

Member Functions

The following functions are members of the CImageWnd Class.

CImageWnd::CimageWnd

```
CImageWnd(
  SapView *pView,
  CWnd *pViewWnd,
  CScrollBar *pHorzScr,
  CScrollBar *pVertScr,
  CWnd *pAppWnd = NULL
);
CImageWnd(
  SapView *pView,
  CWnd *pAppWnd
);
```

Parameters

pView	SapView object to control
pViewWnd	Existing view window object in a dialog-based application (CWnd is defined in MFC)
pHorzScr	Existing horizontal scroll bar object in a dialog-based application (CScrollBar is defined in MFC)
pVertScr	Existing vertical scroll bar object in a dialog-based application (CScrollBar is defined in MFC)
pAppWnd	Application main window object (CWnd is defined in MFC)

Remarks

If your application is dialog-based, you should use the first form of the CImageWnd constructor. The application must already include view window and scroll bar objects, since they are needed by the constructor. If you supply a NULL value for the *pAppWnd* argument, then the value of *pViewWnd* will be used as the main window.

If your application is Single Document Interface (SDI) or Multiple Document Interface (MDI) based, then you should use the second form of the constructor. In this case, the view window is the main application window itself. The scroll bars are then accessed through this window.

CImageWnd | CImageWnd Class Members | GUI Class Hierarchy Chart

CImageWnd::OnMove

void **OnMove**();

Remarks

Adjusts the position of the view window following a WM_MOVE message. You must call OnMove from within your application's WM_MOVE message handler.

CImageWnd | CImageWnd Class Members | GUI Class Hierarchy Chart

CImageWnd::OnPaint

void **OnPaint**();

Remarks

Repaints uncovered areas of the view window following a WM_PAINT message. You must call OnPaint from within your application's WM_PAINT message handler.

CImageWnd | CImageWnd Class Members | GUI Class Hierarchy Chart

CImageWnd::OnSize

void **OnSize**();

Remarks

Adjusts the size of the view window following a WM_SIZE message. You must call OnSize from within your application's WM_SIZE message handler.

In a dialog-based application, the view window size and the scroll bars are adjusted to reach the bottom right corner of the main application window. Therefore, do not place any controls at the right or at the bottom of the view window, as they will be hidden when running the application.

CImageWnd | CImageWnd Class Members | GUI Class Hierarchy Chart

CImageWnd::OnHScroll

void **OnHScroll**(UINT *nSBCode*, UINT *nPos*);

Parameters

nSBCode	Scroll-bar code that indicates the scrolling request
nPos	Horizontal scroll-bar position

Remarks

Adjusts the horizontal scroll bar following a WM_HSCROLL message. You must call OnHScroll from within your application's WM_HSCROLL message handler, using the *nSBCode* and *nPos* arguments you received in this handler.

CImageWnd | CImageWnd Class Members | GUI Class Hierarchy Chart

CImageWnd::OnVScroll

void **OnVScroll**(UINT *nSBCode*, UINT *nPos*);

Parameters

nSBCode	Scroll-bar code that indicates the scrolling request
nPos	Vertical scroll-bar position

Remarks

Adjusts the vertical scroll bar following a WM_VSCROLL message. You must call OnVScroll from within your application's WM_VSCROLL message handler, using the *nSBCode* and *nPos* arguments you received in this handler.

CImageWnd | CImageWnd Class Members | GUI Class Hierarchy Chart

CImageWnd::OnLButtonDown

BOOL **OnLButtonDown**(CPoint *point*);

Parameters

point Mouse position coordinate (CPoint is defined in MFC)

Return Value

Returns TRUE if *point* is inside the view window, FALSE otherwise

Remarks

Begins or ends drawing of the ROI tracker following a WM_LBUTTONDOWN message. You must call OnLButtonDown from within your application's WM_LBUTTONDOWN message handler.

If the mouse position is outside the view window, then the ROI tracker rectangle is clipped to the window.

CImageWnd | CImageWnd Class Members | GUI Class Hierarchy Chart

CImageWnd::OnSetCursor

BOOL **OnSetCursor**(UINT *nHitTest*);

Parameters

nHitTest Hit test code

Return Value

Returns TRUE if the ROI tracker is active and the cursor position is inside its rectangle, FALSE otherwise

Remarks

Sets the cursor shape according to the position of the mouse over the ROI tracker following a WM_SETCURSOR message. You must call OnSetCursor from within your application's WM_SETCURSOR message handler, using the *nHitTest* argument you received in this handler.

The cursor shape indicates whether you can resize or move the rectangle.

CImageWnd | CImageWnd Class Members | GUI Class Hierarchy Chart

CImageWnd:: IsRoiTrackerActive

BOOL **IsRoiTrackerActive**();

Remarks

Returns TRUE if the ROI tracker is currently displayed, FALSE otherwise.

To activate the ROI tracker, click once in the view window, then keep the left mouse button down while dragging. Release the left button when done.

CImageWnd | CImageWnd Class Members | GUI Class Hierarchy Chart

CImageWnd::DisplayRoiTracker

void **DisplayRoiTracker**();

Remarks

Redraws the ROI tracker (clipped to the view window) if it is currently active

CImageWnd | CImageWnd Class Members | GUI Class Hierarchy Chart

CImageWnd::HideRoiTracker

void **HideRoiTracker**();

Remarks

Hides the ROI tracker if it is currently active

CImageWnd | CImageWnd Class Members | GUI Class Hierarchy Chart

CImageWnd::GetSelectedRoi

CRect **GetSelectedRoi**();

Remarks

Gets the rectangle corresponding to the ROI tracker (CRect is defined in MFC)

CImageWnd | CImageWnd Class Members | GUI Class Hierarchy Chart

CImageWnd::SelectRoi

void **SelectRoi**(CRect *rect*);

Parameters

rect New tracker ROI rectangle (CRect is defined in MFC)

Remarks

Sets the ROI tracker rectangle. You must call DisplayRoiTracker after SelectRoi in order to draw the ROI.

CImageWnd | CImageWnd Class Members | GUI Class Hierarchy Chart

CImageWnd::IsPointInside

BOOL **IsPointInside**(CPoint *point*);

Parameters

point Coordinates of the mouse position in the application window (CPoint is defined in MFC)

Remarks

Returns TRUE if the specified point is inside the view window, FALSE otherwise

CImageWnd | CImageWnd Class Members | GUI Class Hierarchy Chart

CImageWnd::TranslateMousePos

CPoint **TranslateMousePos**(CPoint *point*);

Parameters

point Coordinates of the mouse position in the application window (CPoint is defined in MFC)

Return Value

Translated point coordinate

Remarks

Converts mouse position coordinates to the view window coordinate system. The current scrolling position is taken into account.

CImageWnd | CImageWnd Class Members | GUI Class Hierarchy Chart

CImageWnd::Invalidate

void **Invalidate**(BOOL *bErase* = TRUE);

Parameters

bErase TRUE to also erase the view window background, FALSE otherwise

Remarks

Invalidates the view window client area so that it will be completely redrawn through the OnPaint method

CImageWnd | CImageWnd Class Members | GUI Class Hierarchy Chart

CImageWnd::UpdateClientArea

void **UpdateClientArea**();

void **UpdateClientArea**(COLORREF *color*);

Parameters

color New view window background color (COLORREF is defined in GDI)

Remarks

Repaints unused regions of the view window client area using either the current background color, or the specified *color* argument. This is useful when the displayed image is smaller than the view window.

CImageWnd | CImageWnd Class Members | GUI Class Hierarchy Chart

CImageWnd::GetPixelString

CString **GetPixelString**(CPoint *point*);

Parameters

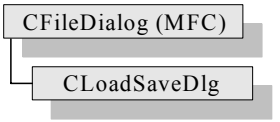
point Coordinates of the mouse position in the application window (CPoint is defined in MFC)

Remarks

Gets a text description of the pixel value a specific position in the view window (CString is defined in MFC). If the point is outside the view window, the string is empty.

CImageWnd | CImageWnd Class Members | GUI Class Hierarchy Chart

CLoadSaveDlg



The CLoadSaveDlg Class allows you to load/save an image file from/to a SapBuffer object. You may create a modal version of this dialog after calling SapBuffer::Create, except if you do not supply a SapBuffer object in the constructor.

Depending on the file type, some additional dialog classes are automatically used to specify additional type-specific options:

- CAviFileDlg (AVI files)
- CJpegFileDlg (JPEG files)
- CTiffFileDlg (TIFF files)
- CRawFileDlg (RAW files)

#include <SapClassGui.h>

CLoadSaveDlg Class Members | GUI Class Hierarchy Chart

CLoadSaveDlg Class Members

Construction	
CLoadSaveDlg	Class constructor
Attributes	
GetPathName	Get the full path name of the file to load from or save to
GetOptions	Gets a text string with the load/save options
GetStartFrame	Gets the first frame number to be loaded in a sequenced image file

CLoadSaveDlg | GUI Class Hierarchy Chart

Member Functions

The following functions are members of the CLoadSaveDlg Class.

CLoadSaveDlg::CLoadSaveDlg

CLoadSaveDlg(
  CWnd* *pParent*,
  SapBuffer **pBuffer*,
  BOOL *isFileOpen* = TRUE,
  BOOL *isSequence* = FALSE
);

Parameters

pParent	Pointer to the parent window of this dialog (CWnd is defined in MFC)
pBuffer	Pointer to a SapBuffer object used for loading/saving the image file (can be NULL)
isFileOpen	TRUE to create a 'File | Load' dialog box, FALSE to create a 'File | Save' dialog box
isSequence	TRUE to include sequenced file types in the filter list (for example, AVI), FALSE to include basic file types (for example, BMP and JPEG).

Remarks

The CLoadSaveDlg constructor does not immediately show the dialog. This happens only when you call its DoModal method.

There are two ways to use this dialog. The first is to set the *pBuffer* argument to a non-NULL value. The selected image file is then automatically loaded from / saved to the SapBuffer object. This means that you must have called SapBuffer::Create first. This usage is the simpler of the two. However, when loading an image the buffer may not necessarily be compatible with the selected image file causing an automatic data conversion to be performed (for example, trying to load a color image file into a monochrome buffer).

The second way is to set *pBuffer* to NULL. You must then explicitly load/save the image file by calling the SapBuffer::Load and SapBuffer::Save methods after the dialog is closed. You may obtain the necessary arguments from the GetPathName, GetOptions, and GetStartFrame methods. When loading, this allows you to make your buffer compatible with the image file by using either the SapBuffer::SapBuffer(**filename*) constructor or the SapBuffer::SetParametersFromFile method before calling SapBuffer::Create.

Example 1: Loading a non-sequenced image file (in an existing buffer)

```
CLoadSaveDlg dlg(this, m_Buf, TRUE);
dlg.DoModal();
```

Example 2: Loading a non-sequenced image file (in a non-existing buffer)

```
CLoadSaveDlg dlg(this, NULL, TRUE);
```

```
dlg.DoModal();

SapBuffer buf(dlg.GetPathName());   // Constructs a compatible buffer
buf.Create();
buf.Load(dlg.GetPathName(), dlg.GetOptions());
```

Example 3: Loading a sequenced image file (in an existing buffer)

```
CLoadSaveDlg dlg(this, m_Buf, TRUE, TRUE);
dlg.DoModal();
```

Example 4: Loading a sequenced image file (in a non-existing buffer)

```
CLoadSaveDlg dlg(this, NULL, TRUE, TRUE);
dlg.DoModal();

SapBuffer buf(dlg.GetPathName());   // Constructs a compatible buffer
buf.Create();
buf.Load(dlg.GetPathName(), dlg.GetOptions(),dlg.GetStartFrame());
```

Example 5: Saving a non-sequenced image file

```
CLoadSaveDlg dlg(this, m_Buf, FALSE);
dlg.DoModal();
```

Example 6: Saving a sequenced image file

```
CLoadSaveDlg dlg(this, m_Buf, FALSE, TRUE);
dlg.DoModal();
```

CLoadSaveDlg | CLoadSaveDlg Class Members | GUI Class Hierarchy Chart

CLoadSaveDlg::GetPathName

CString **GetPathName**();

Remarks

Get the full path name of the file to load from / save to. When no SapBuffer object is specified in the CLoadSaveDlg constructor, you need to use GetPathName before SapBuffer::Load or SapBuffer::Save.

CLoadSaveDlg | CLoadSaveDlg Class Members | GUI Class Hierarchy Chart

CLoadSaveDlg::GetOptions

CString **GetOptions**();

Remarks

Gets a text string with the load/save options. When no SapBuffer object is specified in the CLoadSaveDlg constructor, you need to use GetOptions before SapBuffer::Load or SapBuffer::Save.

CLoadSaveDlg | CLoadSaveDlg Class Members | GUI Class Hierarchy Chart

CLoadSaveDlg::GetStartFrame

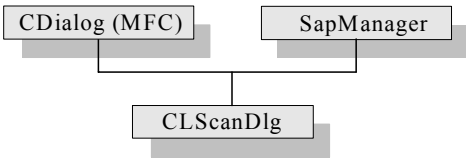
int **GetStartFrame**();

Remarks

Gets the first frame number to be loaded in a sequenced image file. When no SapBuffer object is specified in the CLoadSaveDlg constructor, you need to use GetOptions before SapBuffer::Load or SapBuffer::Save.

CLoadSaveDlg | CLoadSaveDlg Class Members | GUI Class Hierarchy Chart

CLScanDlg



The CLScanDlg Class allows you to dynamically adjust the following acquisition parameters related to linescan cameras:

- Line triggering options
- Frame triggering options
- Shaft encoder options
- Linescan direction options

You may create a modal version of this dialog after calling SapAcquisition::Create. Changes made through this dialog are saved in the current SapAcquisition object only. You must call the SapAcquisition::SaveParameters method if you need to save the new values to an acquisition configuration file (CCF). Otherwise, reloading a new CCF file (through the CAcqConfigDlg Class) or calling SapAcquisition::Destroy causes all changes to be lost.

#include <SapClassGui.h>

CLScanDlg Class Members | Base Class: SapManager | GUI Class Hierarchy Chart

CLScanDlg Class Members

Construction

CLScanDlg	Class constructor

CLScanDlg | Base Class: SapManager | GUI Class Hierarchy Chart

Member Functions

The following functions are members of the CLScanDlg Class.

CLScanDlg::CLScanDlg

CLScanDlg(
  CWnd* *pParent*,
  SapAcquisition **pAcq*
);

Parameters

pParent	Pointer to the parent window of this dialog (CWnd is defined in MFC)
pAcq	Pointer to the related SapAcquisition object

Remarks

The CLScanDlg constructor does not immediately show the dialog. This happens only when you call its DoModal method.

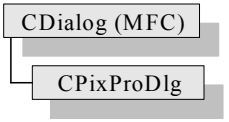
You can only call DoModal after SapAcquisition::Create. Also, you must call SapAcquisition::SaveParameters after using the dialog if you want to save your changes to an acquisition configuration file (CCF).

Example

```
CLScanDlg dlg(this, m_Acq);
dlg.DoModal();
```

CLScanDlg | CLScanDlg Class Members | GUI Class Hierarchy Chart

CPixProDlg



The CPixProDlg Class allows you to modify the parameters of a SapPixPro object.

You may create a modal version of this dialog after calling SapPixPro::Create, except if you rely on CPixProDlg to automatically create a SapPixPro object.

Depending on the pixel processor design, some additional dialog classes are automatically used to specify additional design-specific options:

- CPixProAbsSubDlg
- CPixProIntScaleDlg
- CPixProRefImageDlg
- CPixProUflowSubDlg
- CPixProWAvgDlg

#include <SapClassGui.h>

CPixProDlg Class Members | GUI Class Hierarchy Chart

CPixProDlg Class Members

Construction	
CPixProDlg	Class constructor
Attributes	
GetPixPro	Gets the original or automatically created SapPixPro object

CPixProDlg | GUI Class Hierarchy Chart

Member Functions

The following functions are members of the CPixProDlg Class.

CPixProDlg::CPixProDlg

CPixProDlg(
  CWnd* *pParent*,
  SapPixPro **pPixPro*,
  SapXferNode **pSrcNode* = NULL
);

Parameters

pParent	Pointer to the parent window of this dialog (CWnd is defined in MFC)
pPixPro	Pointer to the related SapPixPro object
pSrcNode	SapXferNode object that provides pixel processor parameters if *pPixPro* is NULL

Remarks

The CPixProDlg constructor does not immediately show the dialog. This happens only when you call its DoModal method.

You may specify an existing SapPixPro object through the *pPixPro* argument. Alternatively, you may set *pPixPro* to NULL and specify an existing SapXferNode object (usually SapAcquisition). In this case, a new SapPixPro object is automatically created using the same SapLocation attribute as *pSrcNode*.

You can only call DoModal after SapPixPro::Create (except of *pPixPro* is NULL) and before SapTransfer::Create.

Example

```
// Both m_PixPro (SapPixPro) and m_Acq (SapAcquisition) are used.
// The first time, m_PixPro is NULL and m_Acq is used for initialization.
// The second time, m_PixPro is used for initialization.

CPixProDlg dlg(this, m_PixPro, m_Acq);
if (dlg.DoModal() == IDOK)
{
       // Destroy all objects (including m_PixPro)
       DestroyObjects();


SapPixPro pixpro;

   if (!m_PixPro)
   {
      // Allocate new object
      m_PixPro = new SapPixPro(dlg.GetPixPro());
```

```
    }
    else
    {
       // Create a backup object
       pixpro = *m_PixPro;

        // Update current object
                *m_PixPro = dlg.GetPixPro();
    }


        // Recreate all objects (including m_PixPro)
    if (!CreateObjects())
    {
       // If an error occurred, retrieve the backup copy and try again
       *m_PixPro = pixpro;
       if (!CreateObjects())
       {
           delete m_PixPro;
           m_PixPro = NULL;
           CreateObjects();
       }
    }
}
```

CPixProDlg | CPixProDlg Class Members | GUI Class Hierarchy Chart

CPixProDlg::GetPixPro

SapPixPro &**GetPixPro**();

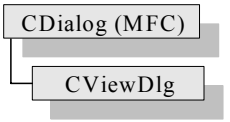
Remarks

If you previously called the CPixProDlg constructor with a non-NULL *pPixPro* argument, this method returns the original SapPixPro object.

If you called the constructor with *pPixPro* equal to NULL, but you specified an appropriate value for the source node, then this method returns a new SapPixPro object automatically created using the same SapLocation attribute as *pSrcNode*.

CPixProDlg | CPixProDlg Class Members | GUI Class Hierarchy Chart

CViewDlg



The CViewDlg Class allows you to dynamically adjust the following view parameters:

- Viewing range value

You may create a modal version of this dialog after calling SapView::Create.

#include <SapClassGui.h>

CViewDlg Class Members | GUI Class Hierarchy Chart

CViewDlg Class Members

Construction

CViewDlg	Class constructor

CViewDlg | GUI Class Hierarchy Chart

Member Functions

The following functions are members of the CViewDlg Class.

CViewDlg::CViewDlg

```
CViewDlg(
  CWnd *pParent,
  SapView *pView
);
```

Parameters

pParent	Pointer to the parent window of this dialog (CWnd is defined in MFC)
pView	Pointer to the related SapView object

Remarks

The CViewDlg constructor does not immediately show the dialog. This happens only when you call its DoModal method. You can only call DoModal after SapView::Create.

Example

```
CViewDlg dlg(this, m_View);
dlg.DoModal();
```

CViewDlg | CViewDlg Class Members | GUI Class Hierarchy Chart

DALSA Contact Information

Sales Information

Visit our web site:	http://www.imaging.com
Email:	info@dalsa-coreco.com

International/Canada

DALSA
7075 Place Robert-Joncas
Suite 142
St. Laurent, Quebec, Canada
H4M 2Z2
Tel:(514) 333-1301
Fax:(514) 333-1388

USA

DALSA
Building 8, Floor 2
900 Middlesex Turnpike
Billerica, MA 01821

Tel:(978) 670-2000
Fax:(978) 670-2010

Technical Support

Any support question or request can be submitted via our web site:

Technical support form via our web page: Support requests for imaging product installations, Support requests for imaging applications	http://www.imaging.com/support
Camera support information	http://www.imaging.com/camsearch
Product literature and driver updates	http://www.imaging.com/download

When encountering hardware or software problems, please have the following documents included in your support request:

1. The DALSA Diagnostic Tool XML report file
2. The DALSA Log Viewer .txt file
3. The PCI Diagnostic PciDump.txt file
4. The DALSA Device Manager BoardInfo.txt file

Note, all these tools are available from the Windows start menu shortcut **Start•Programs•DALSA•Sapera LT•Tools**.

Glossary of Terms

Composite Video

A single signal that includes both color video and timing information. NTSC and PAL are composite video standards, while RGB is not.

DLL

Dynamic Link Library. The supplied DLLs form the software interface between a Windows application and the DALSA Coreco hardware device.

Element

A data unit within the buffer, which may or may not be a pixel.

Interlaced

The standard television method of raster scanning, in which the image is the product of two fields, each of which is made up of the image's alternate lines (that is, one field is comprised of lines 1, 3, 5, etc., and the other is comprised of lines 2, 4, 6, etc.)

Monochrome

A video source with only one component, usually meant to refer to a black-and-white composite signal. A monochrome composite video source has no chroma information.

Pixel

A single picture element, the smallest individual digital video component The number of pixels describes the number of digital samples taken of the analog video signal. The number of pixels per video line by the number of active video lines describes the acquisition image resolution. The binary size of each pixel (that is, 8 bits, 15 bits, 24 bits) defines the number of gray levels or colors possible for each pixel.

RGB

Red, Green, Blue. Commonly used to refer to a non-composite video standard which uses these three colors in combination to generate a color video image.

Sync

The basic piece of information which tells a video display (TV or computer monitor) where to put the picture. Horizontal sync, or HSYNC, controls the left-right dimension and vertical sync, or VSYNC controls the top-to-bottom dimension.

Video Input Conditioning , VIC

The act of modifying an analog video signal via bandwidth filtering or gain amplification.

YUV

A common color space used in composite video color systems. Y is the luma component while U and V are the color difference components.

Index

A

Abort
  SapTransfer 301
AddPair
  SapTransfer 291
API 7
ApplyLut
  SapAcquisition 32
  SapView 332
Arithmetic
  SapLut 231
AutoTrigger
  SapGio 201

B

Basic classes 7, 12, 13, 14
BinaryPattern
  SapLut 232
Boolean 224, 232, 241
  SapLut 232
Borland C++ Builder 8

C

CAB 21, 80, 131, 132, 133, 134, 157, 266, 377
CAbortDlg
  CAbortDlg 364
CAcqConfigDlg
  CAcqConfigDlg 366
CAcqConfigDlg Class 404
CAcqDlg
  CAcqDlg 369
CanEnableLut
  SapAcquisition 29
CAScanDlg
  CAScanDlg 371
CBayerOptionsDlg
  CBayerOptionsDlg 372
CBufDlg
  CBufDlg 375
CCabDlg
  CCabDlg 378
CCF 21, 22, 23, 26, 31, 43, 44, 45, 365, 366, 367, 369, 371, 381, 404, 405
CCompDlg
  CCompDlg 381
CDspDlg
  CDspDlg 383
CFlatFieldDlg
  CFlatFieldDlg 386
CGioInputDlg
  CGioInputDlg 388
CGioOutputDlg
  CGioOutputDlg 390
CImageWnd
  CimageWnd 393
Circle
  SapGraphic 213
class library 7
Clear
  SapBuffer 96
  SapFlatField 188
  SapGraphic 213
CLoadSaveDlg
  CLoadSaveDlg 401
CLScanDlg
  CLScanDlg 405
COLORREF 399
ComputeGain

SapFlatField 188
ComputeOffset
SapFlatField 188
Connect
SapTransfer 297
container 144
Convert
SapBayer 77
Copy
SapBuffer 99
SapLut 237
CopyAll
SapBuffer 99
CPixProDlg
CPixProDlg 407
Create
SapAcqDevice 45
SapAcquisition 26
SapBayer 69
SapBuffer 88
SapBufferRemote 120
SapBufferRoi 123
SapBufferWithTrash 129
SapCab 133
SapCounter 137
SapDisplay 149
SapDsp 159
SapFeature 162
SapFlatField 180
SapGio 195
SapGraphic 209
SapLut 226
SapPixPro 268
SapProcessing 279
SapTransfer 290
SapView 319
CustomCommand
SapAcquisition 38
CViewDlg
CViewDlg 409
CWnd 366

D

D suffix files 11
DALSA Log Viewer 242
Data Classes
SapData 15
SapDataFloat 19
SapDataFPoint 20
SapDataFRGB 20
SapDataHSI 18
SapDataHSV 18
SapDataMono 16
SapDataPoint 19
SapDataRGB 16
SapDataRGBA 17
SapDataYUV 17
data types 15
demos and examples 14
Destroy
SapAcqDevice 45
SapAcquisition 26
SapBayer 69
SapBuffer 88
SapBufferRemote 120
SapBufferRoi 123
SapBufferWithTrash 130
SapCab 133
SapCounter 137
SapDisplay 149
SapDsp 159
SapFeature 162
SapFlatField 180
SapGio 195
SapGraphic 209
SapLut 227
SapPixPro 268
SapProcessing 280
SapTransfer 290

SapView 320
DisableCallback
SapCounter 142
SapGio 201
Disconnect
SapTransfer 298
DisplayMessage
SapManager 255
DisplayRoiTracker
CimageWnd 397
Dot
SapGraphic 214
DSP 21, 80, 131, 157, 158, 159, 309, 312, 382, 383, 384
dynamic-link libraries (DLLs) 8, 11

E

Ellipse
SapGraphic 214
Enable
SapBayer 76
EnableCallback
SapCounter 142
SapGio 200
Enabled
SapFlatField 189
EnableLut
SapAcquisition 33
SapBayer 79
EnablePixelReplacement
SapFlatField 190
Execute
SapFlatField 190
SapProcessing 283
ExecuteNext
SapProcessing 284

F

File Locations 8
Flush
SapGraphic 218
Freeze
SapTransfer 300

G

Gamma
SapLut 233
GDI 7, 207, 218, 219, 316, 318, 321, 399
general I/O 135, 193
GetAccessMode
SapFeature 163
GetAcqDevice
SapAcqDeviceCallbackInfo 60
GetAcquisition
CAcqConfigDlg 367
SapAcqCallbackInfo 41
GetAcquisition, SetAcquisition
SapBayer 69
SapFlatField 181
GetAddress
SapBuffer 112
GetAlign, SetAlign
SapBayer 72
GetAuxiliaryTimeStamp
SapAcqDeviceCallbackInfo 61
GetAvailAlign
SapBayer 72
GetAvailMethod
SapBayer 73
GetAvailPinConfig
SapGio 197
GetAverage
SapFlatField 191
GetBackColor, SetBackColor
SapGraphic 211

GetBankId, SetBankId
SapPixProParams 275
GetBaseUnits, SetBaseUnits
SapCounter 139
GetBayerBuffer
SapBayer 70
GetBayerBufferCount
SapBayer, SetBayerBufferCount
SapBayer 71
GetBuffer
CBufDlg 376
GetBuffer, SetBuffer
SapBayer 70
SapFlatField 181
SapProcessing 280
SapView 320
GetBufferOffset, GetBufferGain
SapFlatField 181
GetBufName
SapBuffer 94
GetBytesPerPixel
SapBuffer 93
GetCab
CCabDlg 379
GetCallback
SapAcquisition 28
SapCounter 138
SapGio 196
SapProcessing 281
SapTransfer 293
SapView 322
SapXferPair 346
GetCamSel, SetCamSel
SapAcquisition 28
GetCapability
SapAcquisition 37
SapBuffer 116
SapCab 133
SapCounter 142
SapDisplay 156
SapGio 203
SapGraphic 218
SapLut 238
SapPixPro 271
SapTransfer 303
SapView 334
GetCategory
SapFeature 164
GetColor, SetColor
SapGraphic 211
GetConfigFile, SetConfigFile
SapAcqDevice 45
SapAcquisition 26
GetConfigIndex, SetConfigIndex
SapPixProParams 274
GetConfigInfo
SapPixPro 271
GetConfigName, SetConfigName
SapAcqDevice 46
GetContext
SapAcqCallbackInfo 41
SapAcqDeviceCallbackInfo 61
SapAcquisition 28
SapCounter 138
SapCounterCallbackInfo 146
SapGio 196
SapGioCallbackInfo 206
SapManCallbackInfo 262
SapProCallbackInfo 286
SapProcessing 281
SapTransfer 294
SapView 322
SapViewCallbackInfo 337
SapXferCallbackInfo 340
SapXferPair 346
GetCorrectionType, SetCorrectionType
SapFlatField 182
GetCount
SapCounter 140
GetCount, SetCount

SapBuffer 88
GetCounter
SapCounterCallbackInfo 145
GetCounterStamp
SapBuffer 114
GetCounterStampInfo
SapTransfer 296
GetCounterStampTimeBase, SetCounterStampTimeBase
SapXferPair 350
GetCustomData
SapAcqDeviceCallbackInfo 62
GetCustomSize
SapAcqDeviceCallbackInfo 62
GetCycleMode, SetCycleMode
SapXferPair 348
GetDC
SapDisplay 153
SapView 331
GetDetectType, SetDetectType
SapCounter 139
GetDeviationMaxBlack, SetDeviationMaxBlack
SapFlatField 183
GetDeviationMaxWhite, SetDeviationMaxWhite
SapFlatField 184
GetDirection, SetDirection
SapCounter 140
GetDisplay, SetDisplay
SapView 321
GetDisplayName
SapFeature 164
GetDisplayStatusMode, SetDisplayStatusMode
SapManager 242
GetDrawMode, SetDrawMode
SapGraphic 210
GetDsp
CDspDlg 384
GetDst
SapXferPair 345
GetDstPort
SapXferPair 345
GetElementSize
SapLut 228
GetEnumCount
SapFeature 165
GetEnumString
SapFeature 165
GetEnumStringFromValue
SapFeature 165
GetEnumValue
SapFeature 166
GetEnumValueFromString
SapFeature 166
GetErrorMessage
SapManCallbackInfo 263
GetErrorValue
SapManCallbackInfo 263
GetEventCount
SapAcqCallbackInfo 41
SapAcqDevice 46
SapAcqDeviceCallbackInfo 63
SapCounterCallbackInfo 146
SapGioCallbackInfo 206
SapXferCallbackInfo 340
GetEventCountSource, SetEventCountSource
SapXferPair 348
GetEventIndex
SapAcqDeviceCallbackInfo 63
GetEventIndexByName
SapAcqDevice 47
GetEventInfo
SapAcqDeviceCallbackInfo 63
GetEventNameByIndex
SapAcqDevice 47
GetEventType
SapAcqCallbackInfo 41

SapCounterCallbackInfo 146
SapGioCallbackInfo 206
SapXferCallbackInfo 340
GetEventType, SetEventType
SapAcquisition 27
SapXferPair 347
GetFeatureCount
SapAcqDevice 48
GetFeatureIndex
SapAcqDeviceCallbackInfo 64
GetFeatureIndexByName
SapAcqDevice 48
GetFeatureInfo
SapAcqDevice 49
GetFeatureNameByIndex
SapAcqDevice 49
GetFeatureValue
SapAcqDevice 50
GetFieldOrder, SetFieldOrder
SapXferParams 355
GetFile, SetFile
SapDsp 159
SapPixPro 268
GetFlatFieldCycleMode, SetFlatFieldCycleMode
SapXferPair 352
GetFlipMode, SetFlipMode
SapAcquisition 29
SapXferPair 349
GetFormat, SetFormat
SapBuffer 90
SapLut 228
SapXferParams 356
GetFormatType
SapManager 253
GetFrameRate, SetFrameRate
SapBuffer 93
GetFramesOnBoard, SetFramesOnBoard
SapXferPair 351
GetFramesPerCallback, SetFramesPerCallback
SapXferPair 351
GetFrameType, SetFrameType
SapXferParams 354
GetFunctionIndex, SetFunctionIndex
SapPixProParams 274
GetFunctionInfo
SapPixPro 270
GetGainDivisor, SetGainDivisor
SapFlatField 184
GetGamma, SetGamma
SapBayer 75
GetGenericParam0..3
SapAcqDeviceCallbackInfo 64
GetGio
SapGioCallbackInfo 205
GetHandle
SapCounter 141
SapDisplay 153
SapFeature 166
SapGio 200
SapGraphic 212
SapLut 229
SapTransfer 297
SapView 329
SapXferNode 361
GetHandle, operator[]
SapBuffer 94
GetHandles
SapBuffer 94
GetHeight
SapDisplay 150
SapView 323
GetHeight, SetHeight
SapBuffer 89
SapBufferRoi 125
SapXferParams 355
GetHostTimeStamp
SapAcqDeviceCallbackInfo 65

GetImmediateMode, SetImmediateMode
SapView 327
GetInc
SapFeature 167
GetIndex
SapProcessing 281
SapView 324
GetIndex, SetIndex
SapBuffer 95
SapFlatField 186
GetInstallDirectory
SapManager 257
GetIntegrateCount, SetIntegrateCount
SapPixProParams 275
GetIntegrateFactor, SetIntegrateFactor
SapPixProParams 276
GetKeyColor, SetKeyColor
SapView 326
GetLabel
SapAcqDevice 51
SapAcquisition 27
GetLastStatus
SapManager 255
GetLocation
SapCounter 137
SapDisplay 149
SapGio 195
SapGraphic 210
SapLut 227
SapTransfer 291
GetLocation, SetLocation
SapFeature 167
SapXferNode 360
GetLut
SapAcquisition 32
SapBayer 79
SapView 332
GetMax
SapFeature 168
GetMethod, SetMethod
SapBayer 73
GetMin
SapFeature 168
GetModeName, SetModeName
SapAcqDevice 51
GetName
SapFeature 169
GetNumConfigs
SapPixPro 270
GetNumEntries, SetNumEntries
SapLut 227
GetNumFlatField, SetNumFlatField
SapFlatField 180
GetNumFunctions
SapPixPro 270
GetNumLinesAverage,
SetNumLinesAverage
SapFlatField 185
GetNumPages
SapLut 228
GetNumPairs
SapTransfer 291
GetNumPins
SapGio 197
GetNumPlanarInputs
SapAcquisition 30
GetOptions
CLoadSaveDlg 403
GetOutputFormat, SetOutputFormat
SapBayer 75
GetOutputXferParams,
SetOutputXferParams
SapPixPro 269
GetOverlayMode, SetOverlayMode
SapView 325
GetPage
SapBuffer 110
GetPair
SapTransfer 292
GetPairIndex

SapXferCallbackInfo 341
GetParameter, SetParameter
SapAcquisition 37
SapBuffer 117
SapCab 134
SapCounter 143
SapDisplay 156
SapGio 203
SapGraphic 219
SapLut 239
SapPixPro 272
SapTransfer 304
SapView 335
GetParameters, SetParameters
SapBuffer 91
SapXferParams 357
GetParams, SetParams
SapPixPro 269
GetParent, SetParent
SapBufferRoi 124
GetPathName
CLoadSaveDlg 402
GetPinConfig, SetPinConfig
SapGio 198
GetPinNumber
SapGioCallbackInfo 206
GetPinState, SetPinState
SapGio 199
GetPitch
SapBuffer 93
GetPixelDepth
SapDisplay 150
GetPixelDepth, GetPixelDepth
SapXferParams 356
GetPixelDepth, SetPixelDepth
SapBuffer 92
GetPixelDepthMin, GetPixelDepthMax
SapManager 243
GetPixelShift, SetPixelShift
SapXferParams 356
GetPixelString
CimageWnd 399
GetPixPro
CPixProDlg 408
GetPlanarInputs
SapAcquisition 30
GetPollingTime
SapFeature 169
GetProcessing
SapProCallbackInfo 286
GetRange, SetRange
SapView 329
GetRangeMinMax
SapView 328
GetReadOnly, SetReadOnly
SapAcqDevice 52
GetRefImage, SetRefImage
SapPixProParams 275
GetRefreshRate
SapDisplay 150
GetRepresentation
SapFeature 170
GetResetTimeout, SetResetTimeout
SapManager 243
GetResourceCount
SapManager 249
GetResourceIndex
SapLocation 222
SapManager 250
GetResourceName
SapManager 250
GetRoot
SapBufferRoi 124
GetScalingMode, SetScalingMode
SapView 326
GetScrollPos
SapView 323
GetScrollRange
SapView 323
GetSelectedRoi

CimageWnd 397
GetSerialPortName
SapAcquisition 31
GetServerCount
SapManager 244
GetServerHandle
SapManager 248
GetServerIndex
SapLocation 222
SapManager 244
SapManCallbackInfo 262
GetServerName
SapLocation 222
SapManager 245
GetServerSerialNumber
SapManager 248
GetServerType
SapManager 246
GetSign
SapFeature 170
GetSignalStatus
SapAcqCallbackInfo 42
SapAcquisition 34
GetSiToNativeExp10
SapFeature 171
GetSiUnit
SapFeature 172
GetSrc
SapXferPair 344
GetSrcIndex
SapXferPair 345
GetSrcNode, SetSrcNode
SapXferNode 360
GetSrcPort
SapXferNode 360
SapXferPair 344
GetStartFrame
CLoadSaveDlg 403
GetStartMode, SetStartMode
SapTransfer 295
GetState
SapBuffer 108
GetStats
SapFlatField 191
GetStringFromFormat
SapManager 254
GetTextAlign, SetTextAlign
SapGraphic 212
GetThreadPriority, SetThreadPriority
SapProcessing 282
SapView 329
GetTime
SapPerformance 265
SapProcessing 281
GetTimeMicro
SapPerformance 265
GetTimeMilli
SapPerformance 265
GetToolTip
SapFeature 172
GetTotalSize
SapLut 229
GetTransfer
SapXferCallbackInfo 340
GetTransparency, SetTransparency
SapGraphic 211
GetTrash
SapBufferRoi 126
SapBufferWithTrash 130
GetTrashCallback
SapTransfer 293
GetTrashType, SetTrashType
SapBufferWithTrash 130
GetType
SapDisplay 151
SapFeature 173
GetType, SetType
SapBuffer 90
GetUpdateFeatureMode,
SetUpdateFeatureMode

SapAcqDevice 52
GetVerticalOffset, SetVerticalOffset
SapFlatField 185
GetVideoType, SetVideoType
SapFlatField 183
GetView
SapViewCallbackInfo 337
GetViewArea
SapView 323
GetVisibility
SapFeature 174
GetWBGain, SetWBGain
SapBayer 74
GetWBOffset, SetWBOffset
SapBayer 74
GetWeightingFactor, SetWeightingFactor
SapPixProParams 275
GetWidth
SapDisplay 150
SapView 322
GetWidth, SetWidth
SapBuffer 89
SapBufferRoi 125
SapXferParams 355
GetWindow, SetWindow
SapView 321
GetWindowTitle, SetWindowTitle
SapView 328
GetWriteMode
SapFeature 174
GetXferParams, SetXferParams
SapXferNode 361
GetXMin, SetXMin
SapBufferRoi 124
GetYMin, SetYMin
SapBufferRoi 125
Grab
SapTransfer 300
GUI classes 7, 12, 13, 14

H

HasRange
SapView 328
header files 11
Hide
SapView 331
HideRoiTracker
CimageWnd 397

I

import libraries 8
Init
SapProcessing 283
SapTransfer 297
SapView 330
Integrate and Scale 273, 275, 276
International/Canada Sales Office 411
Internet 5
Invalidate
CimageWnd 399
IsAcqLut
SapBayer 76
IsAutoConnect, SetAutoConnect
SapTransfer 295
IsAutoEmpty, SetAutoEmpty
SapProcessing 282
SapTransfer 294
SapView 324
IsBayerAvailable
SapAcquisition 31
IsCallbackRegistered
SapAcqDevice 53
IsCapabilityValid
SapAcquisition 36
SapDisplay 155
SapFlatField 202
SapTransfer 302
IsEnabled

SapBayer 71
SapFlatField 181
IsEnumEnabled
SapFeature 175
IsEventAvailable
SapAcqDevice 53
IsFeatureAvailable
SapAcqDevice 54
IsFlatFieldAvailable
SapAcquisition 31
IsGrabbing
SapTransfer 294
IsInterlaced
SapDisplay 151
IsLutEnabled
SapAcquisition 29
SapBayer 76
IsMapped
SapBuffer 114
IsOffscreenAvailable
SapDisplay 152
IsOverlayAvailable
SapDisplay 152
IsParameterValid
SapAcquisition 36
SapDisplay 155
SapFlatField 202
SapTransfer 302
IsPixelReplacement
SapFlatField 182
IsPointInside
CimageWnd 398
IsPrimaryVGABoard
SapDisplay 152
IsRegCallback
SapXferPair 345
IsResourceAvailable
SapManager 251
IsRoiTrackerActive
CimageWnd 397
IsSameLocation
SapManager 253
IsSameServer
SapManager 252
IsSavedToConfigFile, SetSavedToConfigFile
SapFeature 175
IsSignalStatusAvailable
SapAcquisition 33
IsSigned
SapLut 229
IsSoftware
SapBayer 71
SapFlatField 182
IsStandard
SapFeature 176
IsStatusOk
SapManager 254
IsSystemLocation
SapManager 252
IsTrash
SapXferCallbackInfo 341
IsUnknown
SapLocation 222

L

libraries 11
Line
SapGraphic 215
Load
SapBuffer 97
SapFlatField 187
SapLut 237
LoadFeatures
SapAcqDevice 54
LUT 223, 372

M

Master mode 370
MergeComponents
  SapBuffer 100
message handler 333, 334, 394, 395, 396
MFC 7, 364, 366, 371, 375, 378, 381, 383, 393, 396, 397, 398, 399, 401, 405, 407, 409
Microsoft Visual C++ .NET 8, 13
Microsoft Visual C++ 6.0 8
Multiple Document Interface (MDI) 393

N

Next
  SapBuffer 96
  SapFlatField 186
Normal
  SapLut 233

O

object-oriented 7
OnHScroll
  CimageWnd 395
  SapView 333
OnLButtonDown
  CimageWnd 396
OnMove
  CimageWnd 394
  SapView 333
OnPaint
  CimageWnd 394
  SapView 333
OnSetCursor
  CimageWnd 396
OnSize
  CimageWnd 394
  SapView 333
OnVScroll
  CimageWnd 395
  SapView 334
operator BOOL
  SapManager 241

P

PDF 5
Pixel Processor 21, 42, 80, 131, 267, 268, 270, 271, 272, 273, 275

R

rapid-application development (RAD) 7
Read
  SapBuffer 101
  SapLut 230
ReadDesign
  SapPixPro 269
ReadElement
  SapBuffer 102
ReadLine
  SapBuffer 103
ReadRect
  SapBuffer 104
Rectangle
  SapGraphic 216
Region of Interest (ROI) 121, 123, 126, 372, 391, 398
Register
  SapBuffer 115
RegisterCallback
  SapAcqDevice 55
ReleaseAddress
  SapBuffer 113
ReleaseDC
  SapDisplay 153
  SapView 332
RemoveAllPairs

SapTransfer 292
Reset
SapPerformance 265
ResetIndex
SapBuffer 95
ResetRoi
SapBufferRoi 126
ResetServer
SapManager 256
Reverse
SapLut 234
ROI tracker 392, 396, 397, 398
Roll
SapLut 234
Run
SapProcessing 284

S

SapAcqCallbackInfo
SapAcqCallbackInfo 40
SapAcqDevice
SapAcqDevice 44
SapAcqToBuf Class 305
SapAcqToBufCab
Specialized Transfer Classes 310
SapAcqToBufCabThruDsp
Specialized Transfer Classes 312
SapAcqToBufCabThruPixPro
Specialized Transfer Classes 312
SapAcqToBufThruDsp
Specialized Transfer Classes 309
SapAcqToBufThruPixPro
Specialized Transfer Classes 309
SapAcqToCab
Specialized Transfer Classes 307
SapAcqToCabThruDsp
Specialized Transfer Classes 309
SapAcqToCabThruPixPro
Specialized Transfer Classes 310
SapAcquisition
SapAcquisition 23
SapBayer
SapBayer 68
SapBuffer
SapBuffer 83
SapBuffer Class 135
SapBufferRemote
SapBufferRemote 119
SapBufferRoi
SapBufferRoi 122
SapBufferWithTrash
SapBufferWithTrash 128
SapBufToBuf
Specialized Transfer Classes 307
SapBufToBufCab
Specialized Transfer Classes 311
SapBufToCab
Specialized Transfer Classes 308
SapCab
SapCab 132
SapCabToBuf
Specialized Transfer Classes 307
SapCabToBufCab
Specialized Transfer Classes 311
SapCabToCab
Specialized Transfer Classes 308
SapCounter
SapCounter 136
SapCounter Class 144, 193
SapCounterCallbackInfo
SapCounterCallbackInfo 145
SapDisplay
SapDisplay 148
SapDsp
SapDsp 158
Sapera LT 7, 11, 14, 15
Sapera server 220, 221, 240, 244, 245, 246, 248, 249, 250, 251, 252, 256, 257, 260, 262

SapFeature
SapFeature 162
SapFlatField
SapFlatField 179
SapGio
SapGio 194
SapGio Class 135, 204
SapGioCallbackInfo
SapGioCallbackInfo 205
SapGioCallbackInfo Class 204
SapGraphic
SapGraphic 209
SapLocation
SapLocation 221
SapLut
SapLut 225
SapManager Class 240, 260
SapManCallbackInfo
SapManCallbackInfo 261
SapManCallbackInfo Class 260
SapMultiAcqToBuf
Specialized Transfer Classes 313
SapMultiAcqToBufCab
Specialized Transfer Classes 315
SapMultiAcqToCab
Specialized Transfer Classes 313
SapMultiCabToBuf
Specialized Transfer Classes 314
SapPerformance
SapPerformance 264
SapPerformance Class 264
SapPixPro
SapPixPro 267
SapPixPro Class 266, 273
SapPixProParams
SapPixProParams 274
SapPixProParams Class 273
SapProCallbackInfo
SapProCallbackInfo 286
SapProcessing
SapProcessing 279
SapProcessing Class 80, 264, 277, 285
SapTransfer
SapTransfer 289
SapTransfer Class 127, 305, 338, 342, 353, 358
SapView
SapView 318
SapView Class 80, 336
SapViewCallbackInfo
SapViewCallbackInfo 337
SapXferCallbackInfo::SapXferCallbackInfo 339
SapXferNode
SapXferNode 359
SapXferPair
SapXferPair 343
SapXferPair Class 358
SapXferParams
SapXferParams 354
Save
SapBuffer 98
SapFlatField 187
SapLut 237
SaveFeatures
SapAcqDevice 56
SaveParameters
SapAcquisition 31
SDI 393
Select
SapTransfer 298
SelectRoi
CimageWnd 398
SetAllPage
SapBuffer 111
SetAllState
SapBuffer 110
SetBatchMode
SapGraphic 217
SetCallbackInfo

SapAcquisition 27
SapCounter 138
SapGio 196
SapProcessing 280
SapTransfer 292
SapView 322
SapXferPair 346
SetFeatureValue
SapAcqDevice 56
SetGainMinMax
SapFlatField 185
SetMode
SapDisplay 154
SetOffsetMinMax
SapFlatField 184
SetPage
SapBuffer 111
SetParametersFromFile
SapBuffer 92
SetPhysicalAddress
SapBuffer 91
SetRoi
SapBufferRoi 126
SetState
SapBuffer 109
SetTrashCallbackInfo
SapTransfer 293
SetVirtualAddress
SapBuffer 91
Shift
SapLut 234
Show
SapView 330
ShowNext
SapView 331
Single Document Interface (SDI) 393
Slope
SapLut 235
Snap
SapTransfer 299
software requirements 8
SoftwareTrigger
SapAcquisition 35
source code 5, 7, 12, 13, 14
source transfer node 21, 42, 80, 131, 292, 358, 360
Specialized Transfer Classes 287, 289, 291, 342
Common Constructor Arguments 305
SapAcqToBuf 306
SplitComponents
SapBuffer 100
Start
SapCounter 141
Stop
SapCounter 141

T

Text
SapGraphic 216
Threshold
SapLut 236
transfer mode 377
TranslateMousePos
CimageWnd 398
trash buffer 127, 129, 289, 293
TTL 193

U

Unregister
SapBuffer 116
UnregisterCallback
SapAcqDevice 57
UpdateClientArea
CimageWnd 399
UpdateFeaturesFromDevice
SapAcqDevice 58
UpdateFeaturesToDevice

SapAcqDevice 59
US Sales Office 411

V

VGA board 147
Visual Studio 6.0 12

W

Wait
SapTransfer 301
web site 411
Web site 5
Weighting Average 273, 275
WhiteBalance
SapBayer 78
Windows 2000 ii
Windows NT ii
Windows Registry 382, 383, 384
Windows XP ii
wrapper 118, 119
Write
SapBuffer 105
SapLut 230
WriteElement
SapBuffer 106
WriteLine
SapBuffer 107
WriteRect
SapBuffer 108